Gilvan Justino

DESENVOLVIMENTO DE AMBIENTE COMPUTACIONAL PARA IMPLEMENTAÇÃO DE MÉTODOS DA TEORIA DA RESPOSTA AO ITEM

Florianópolis – SC
2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gilvan Justino

DESENVOLVIMENTO DE AMBIENTE COMPUTACIONAL PARA IMPLEMENTAÇÃO DE MÉTODOS DA TEORIA DA RESPOSTA AO ITEM

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Dalton Francisco de Andrade
Orientador

Florianópolis, Outubro de 2007

DESENVOLVIMENTO DE AMBIENTE COMPUTACIONAL PARA IMPLEMENTAÇÃO DE MÉTODOS DA TEORIA DA RESPOSTA AO ITEM

Gilvan Justino

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Engenharia de Software e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Rogério Cid Bastos, Dr.
Coordenador

Banca Examinadora:

_____

Prof. Dalton Francisco de Andrade, Dr.
Orientador

_____

Prof. Héliton Ribeiro Tavares, Dr.

_____

Prof. Marcelo Menezes Reis, Dr.

_____

Prof. Paulo Justiniano Ribeiro Jr, Dr.

Agradeço a todos os professores deste curso,
especialmente ao prof. Dalton, pelo apoio oferecido
durante o desenvolvimento deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIAÇÕES

GUI	*Graphical User Interface (Interface Gráfica de Usuário)*
TRI	*Teoria da Resposta ao Item*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
SOA	*Service-oriented architecture (Arquitetura Orientada a Serviços)*

RESUMO

Este trabalho propõe o desenvolvimento de um software para utilização da Teoria da Resposta ao Item (TRI). A Teoria da Resposta ao Item é utilizada para mensurar variáveis não observáveis diretamente (variáveis latentes) e está sendo aplicada em várias áreas, como educação, serviço, qualidade de vida etc. A TRI oferece vários benefícios, dentre os quais pode-se citar: a construção de uma escala calibrada para a variável latente e a possibilidade de comparação entre populações diferentes. A teoria fundamenta-se no uso de modelos matemáticos que buscam expressar a relação entre a probabilidade de um indivíduo apresentar uma certa resposta ao item e características do item e a variável latente.

Para promover o desenvolvimento de um software para suportar a TRI, foi criada uma especificação (modelo) para construção de métodos da TRI, que busca padronizar a forma como os pesquisadores constroem métodos para a TRI, afim de facilitar a utilização e implementação de novos métodos. O software proposto é organizado em plug-ins onde cada plug-in contém a uma implementação de um método da Teoria da Resposta ao Item.

Foi adotada a linguagem R para implementação dos algoritmos dos métodos. A definição dos programas e a interface do plug-in são especificadas utilizando-se a linguagem XML. Os plug-ins são constituídos de programas em linguagem R e suas respectivas descrições de interface, conferindo aos plug-ins a capacidade de portabilidade.

Os resultados deste trabalho são: a especificação de um modelo para construção de métodos da TRI, a implementação de um ambiente gráfico que permite o acoplamento de novos métodos que sejam desenvolvidos de acordo com a especificação proposta neste trabalho e a implementação de um método clássico da TRI seguindo esta especificação.

ABSTRACT

This work proposes the development of a software to use Item Response Theory (IRT). Item Response Theory is used to measure variables not directly observed (latent variables) and it is being applied in many areas, as education, service, life quality, etc. IRT offers many benefits, among them can be cited: the build of a calibrated scale to the latent variable and the possibility to compare different populations. This theory is based in mathematical models that try to express the relationship between the probability of an individual to have some response to the item and item characteristics and a latent variable.

To promote the development of a software to support IRT, it was created a specification (model) to build IRT methods, which intend to standarize the way how the researchers build IRT methods, in order to make easier the use and the implementation of new methods. The proposed software is based in plug-ins, where each plug-in has a implementation of a method of Item Response Theory.

It was adopted the R language to build the algorithms of the methods. The programs definition and the plug-in interface are specified using XML language. Plug-ins are constituted of programs in R language and its interface descriptions.

The results of this work are: the specification of a model to build IRT methods, the implementation of a graphical environment which allows the coupling of new IRT methods built according the specification proposed in this work and the implementation of a classical IRT method using this specification.

1 INTRODUÇÃO

1.1 Considerações Iniciais

O uso de ferramentas computacionais para solucionar problemas científicos se tornou popular devido à alta disponibilidade, elevada capacidade de processamento e preços acessíveis dos computadores atuais. Em muitas áreas, o uso de programas de computador se tornou essencial para colocar em prática métodos propostos no passado.

A Teoria da Resposta ao Item (TRI) é certamente uma das áreas de estudo da estatística que necessita do poder de processamento de computadores, pois a TRI exige a execução de algoritmos complexos.

A TRI compreende um conjunto de modelos matemáticos que buscam mensurar variáveis latentes, isto é, variáveis que não são diretamente observáveis, tais como nível de habilidade, nível de satisfação do cliente, etc. Apesar de não ser possível observar diretamente a variável latente, é possível estimá-la partindo de variáveis secundárias observáveis, associadas à variável latente. Geralmente os valores das variáveis secundárias são obtidos a partir da aplicação de um teste composto por questões (ou itens, conforme o vocabulário da TRI).

A TRI, em oposição à Teoria Clássica dos Testes, tem mostrado atingir resultados mais abrangentes. Através da TRI é possível, por exemplo, construir uma escala métrica para a variável latente. Além disso, as propriedades dos itens mantêm-se constantes, independente do grupo de indivíduos que respondeu ao teste, entre outros. Uma área que vem utilizando a TRI com ênfase é a avaliação de desempenho de estudantes. Pode-se citar como exemplo, a aplicação da TRI no Sistema Nacional de Ensino Básico (SAEB) (ANDRADE et al., 2000), que busca avaliar a qualidade do ensino básico brasileiro.

Os modelos matemáticos da TRI são utilizados para descrever a relação entre as respostas dadas pelos indivíduos aos itens e as suas variáveis latentes. Existem diversos modelos matemáticos propostos na literatura, que buscam se adequar às características dos itens empregados no teste. Para que seja possível realizar a análise com a TRI é necessário dispor de um método para computar o ajuste dos dados ao modelo utilizado. Este processo é chamado de estimação ou calibração. O processamento destes métodos exigem a execução de cálculos numéricos complexos.

Atualmente existem poucas ferramentas computacionais para se trabalhar com a TRI, sendo as mais populares Bilog (SSI, 2007), Parscale (SSI, 2007), Multilog (SSI, 2007) e XCalibre (Assessment, 2007). A operação comum para utilizar estes softwares é a seguinte: o usuário submete um arquivo de dados, contendo a relação de itens do teste, o gabarito do teste, a relação de indivíduos que responderam o teste e suas respectivas respostas, e em seguida o usuário fornece alguns parâmetros para configurar o processamento do método. Depois disso, o usuário solicita a execução dos métodos de estimação, a fim de obter os resultados e analisá-los em seguida. Os métodos de estimação fazem uso intensivo do processador. O arquivo com os dados de entrada pode possuir formatos distintos, embora existam algumas convenções.

Todos estes softwares têm em comum o fato de serem ferramentas comerciais, além de serem softwares que permitem apenas a aplicação da TRI para análise de testes, isto é, estes softwares não permitem a construção de novos métodos. Os modelos e métodos utilizados para análise com a TRI estão em constante evolução, representando o foco principal do estudo da TRI. Para criação de novos métodos da Teoria da Resposta ao Item, geralmente os pesquisadores utilizam ambientes de propósito geral, dotados de linguagens de programação científicas, com bibliotecas matemáticas e estatísticas, de tal forma que seja possível codificar os algoritmos necessários. Enquadram-se nesta relação R (R, 2007), S-Plus (Insightful, 2007) e Ox (Jurgen, 2007).

Como não há um ambiente unificado para criar novos algoritmos, os pesquisadores criam os métodos sem utilizar alguma padronização, dificultando a utilização e compartilhamento do método com outros pesquisadores ou com as pessoas que desejam aplicar o método para análise dos seus testes.

Em geral, as principais dificuldades em trabalhar com a TRI são:

- para o usuário quem tem interesse em utilizar soluções gratuitas para usufruir dos recursos que a TRI oferece precisa conhecer o ambiente que hospeda os programas. A interação normalmente é via linha de comando, sem o uso de interfaces apropriadas para lidar com a TRI;
- para que o usuário saiba como executar os programas do método, deve contar com a documentação elaborada pelo autor do método. Desta forma, para tomar conhecimento de quais parâmetros devem ser informados e quais seus pré-requisitos, pode ser necessário analisar os programas fontes;

- como cada pesquisador tem liberdade de criar os programas da forma que desejar, as implementações não utilizam o conceito de encapsulamento, sendo comum ver o uso de variáveis globais;
- a cada novo método implementado, é necessário codificar a interpretação dos arquivos de dados de entrada.

A proposta desta dissertação busca resolver estas deficiências encontradas na construção e utilização de métodos da TRI, através da definição de um modelo para construção dos métodos. O modelo proposto para construção de métodos da TRI pretende uniformizar o meio como os pesquisadores constroem e distribuem os métodos.

Adicionalmente, é apresentado um ambiente gráfico, voltado para processar análise de testes com a TRI. Os métodos desenvolvidos segundo o modelo proposto são tratados como plug-ins para o ambiente gráfico, que permite, portanto, estender suas funcionalidades básicas.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo geral deste trabalho é propor um modelo para construção de métodos da TRI. O modelo deve permitir a construção dos algoritmos de estimação dos parâmetros dos itens e da variável latente e a especificação de ferramentas da TRI. Deve ser desenvolvido um ambiente gráfico que suporte os métodos criados a partir do modelo proposto, isto é, que permita a seleção e utilização dos métodos implementados.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- definir um modelo para construção de métodos da TRI. O modelo deve definir a forma da declaração dos programas para estimação dos parâmetros dos itens e da variável latente. O modelo também deve permitir que seja possível implementar ferramentas para análise dos dados, tal como a Curva Característica do Item e do Teste e a Função de Informação do Item e do Teste;

- desenvolver um ambiente gráfico capaz de trabalhar com os métodos construídos com o modelo proposto, a fim de demonstrar como deve ser a interação entre o *frontend* (interface gráfica do usuário) e os métodos desenvolvidos.

1.3 Metodologia

O desenvolvimento deste trabalho está dividido nas seguintes etapas:

- fundamentação teórica;
- especificação do modelo;
- especificação do software para suportar os métodos desenvolvidos conforme o modelo especificado;
- desenvolvimento de um método segundo o modelo proposto;
- desenvolvimento do ambiente de software;
- conclusões e discussões gerais.

Um levantamento das características de softwares científicos foi elaborado para compreender as propriedades úteis e necessárias para construção de ambientes científicos. Este estudo revelou que uma característica desejável de softwares científicos é permitir que sejam incorporadas novas funcionalidades ao software, sem que seja necessário compilá-lo novamente. A partir daí, foi conduzido um estudo sobre o desenvolvimento de aplicações baseadas em plug-ins.

Na fase de fundamentação teórica também se buscou o entendimento da TRI, a fim de compreender os aspectos envolvidos com a criação de métodos.

Para desenvolvimento do método, utilizou-se a implementação criada por (PINHEIRO, 2006), onde se modificou o código criado para atender à especificação do modelo proposto.

O ambiente gráfico foi construído utilizando-se o ambiente de desenvolvimento *Delphi* (CODEGEAR, 2007). O ambiente *Delphi* permite o desenvolvimento rápido de aplicações usando a linguagem *Delphi*. A linguagem *Delphi* é uma linguagem de alto nível, compilável, fortemente tipada que suporta programação estruturada e orientada a objetos. Alguns dos benefícios da linguagem são: código fácil de entender, compilação rápida e programação modular. Já o ambiente *Delphi* possui um grande número de

componentes reutilizáveis, muitos deles desenvolvidos por terceiros e disponíveis gratuitamente na internet com seu código fonte.

1.4 Trabalhos Correlatos

As iniciativas em se desenvolver software para a Teoria da Resposta ao Item compreendem a construção de bibliotecas ou programas que tratam de uma conjunto limitado de métodos da Teoria da Resposta ao Item. No primeiro caso, os pesquisadores podem aproveitar funções pré-definidas para criar seus próprios métodos, entretanto, não há qualquer especificação sobre como construir métodos. Já no segundo caso, os softwares encontrados não permitem que o pesquisador construa novos métodos, limitando-se aos métodos já existentes e embutidos no software.

Alguns trabalhos estão relacionados a seguir:

- PlotIRT (HILL & LANGER, 2007): trata-se de um pacote contendo funções em linguagem R, construídas especificamente para gerar gráficos da Curva Característica do Item e do Teste e Função de Informação do Item e do Teste. As funções são aplicáveis para um determinado conjunto de modelos: função logística de 2 e 3 parâmetros, modelos de múltipla escolha graduais e nominais. O pacote não lida com estimação dos parâmetros dos itens e da variável latente. O pacote pode ser encontrado em http://www.unc.edu/~dthissen/dl.html;
- (RIZOPOULOS, 2006) apresenta uma biblioteca R com implementação dos modelos de 2 e 3 parâmetros, além do modelo de resposta gradual. A biblioteca permite o cálculo dos parâmetros dos itens e da variável latente;
- PARAM-3PL: é um programa que permite a estimação dos parâmetros dos itens utilizando o modelo de 3 parâmetros. O programa não suporta pacotes e por isso não é possível realizar análises com outros métodos e não é possível construir novos métodos. Ao contrário dos trabalhos anteriores, PARAM-3PL dispõe de uma interface gráfica. Maiores informações podem ser encontradas em (RUDNER, 2005).

1.5 Organização do Trabalho

Este trabalho está organizado em 6 capítulos. O capítulo 1 apresenta o contexto do trabalho, definindo seus objetivos gerais e específicos, metodologia e trabalhos correlatos.

O capítulo 2 apresenta um estudo sobre Software Científico, onde são abordadas as características comuns e desejáveis de ambientes voltados para resolução de problemas científicos. Também é apresentado brevemente a descrição de dois ambientes de softwares populares entre os pesquisadores.

O capítulo 3 mostra o estudo sobre aplicações que permitem estender suas funcionalidades através do acoplamento de plug-ins.

No capítulo 4 pode ser encontrado um estudo da Teoria da Resposta ao Item, onde são apresentados alguns dos modelos mais conhecidos e as ferramentas da TRI. O modelo para construção de métodos da TRI e o software proposto foram especificados para permitir a criação de tais ferramentas.

O desenvolvimento do trabalho está descrito no capítulo 5. Constam neste capítulo a especificação do modelo para construção dos métodos da TRI e a especificação do ambiente gráfico construído.

O capítulo 6 contém as conclusões e observações finais, além de sugestões para trabalhos futuros.

Ao final, são apresentadas as referências bibliográficas e os anexos.

2 SOFTWARE CIENTÍFICO

Denomina-se computação científica a área que estuda a construção de modelos matemáticos para analisar e resolver problemas científicos e de engenharia com auxílio do computador.

Um modelo matemático é um conjunto de expressões matemáticas que são utilizadas para explicar o comportamento de um determinado sistema em particular.

(FRITZON, 2004) considera um sistema como sendo um componente ou coleção de componentes cujas propriedades se deseja estudar. Os sistemas são denominados sistemas naturais quando envolvem componentes da natureza ou quando ocorrem naturalmente, como por exemplo, a erupção de um vulcão. Já os sistemas artificiais são constituídos de componentes artificiais, como por exemplo, o sistema de telefonia celular. Alguns sistemas são compostos por componentes da natureza e componentes artificiais, neste caso, pode-se tomar como exemplo um sistema de aquecimento solar, que é composto pelos raios solares, nuvens, placas coletoras e reservatório térmico.

Independente de sua classificação, um sistema é descrito em termos de entradas e saídas, isto é, um conjunto de entradas ao ser submetido ao sistema, provoca a geração de um conjunto de saídas. Desta forma, as entradas de um sistema são variáveis do ambiente que influenciam o comportamento do sistema, enquanto que as saídas do sistema são variáveis que são determinadas pelo sistema e que podem eventualmente influenciar o seu ambiente ao redor.

Quando as variáveis de entrada podem ser deduzidas a partir das variáveis de saída, diz-se que o sistema é observável. Se as variáveis de entrada puderem ser manipuladas a fim de se produzir uma saída específica no sistema, concluí-se que o sistema é controlável. Estas propriedades permitem que um sistema possa ser expresso através de um modelo matemático. Uma vez construído o modelo matemático é possível conduzir experimentos para simular situações reais que muitas vezes seriam impossíveis de se reproduzir no mundo real ou até mesmo, muito caras para serem realizadas. O uso do computador para manipular modelos matemáticos torna-se indispensável pois geralmente os modelos exigem algoritmos complexos para processá-los. Sendo assim, o processamento através do computador permite investigar cenários em tempo e custo relativamente razoáveis.

Alguns exemplos de modelos matemáticos são (TREFETHEN et al., 1996) e (ETTER, 2006):

- a construção de modelos para entendimento de efeitos de eventos naturais como terremotos, tsunamis e outros desastres naturais;
- modelos para compreensão de sistemas biológicos através da organização de células em estruturas;
- modelo para prever o crescimento populacional;
- modelos para determinar o comportamento de consumidores.

O uso do computador permite também obter soluções de forma mais segura. Por exemplo, a utilização do computador para construção de um protótipo virtual para análise dos efeitos de colisões em um determinado modelo de automóvel não tem os perigos que se teria com a utilização de protótipos reais (HEATH, 2002).

Devido aos benefícios citados, muitas nações têm considerada a computação científica como peça fundamental para o crescimento tecnológico pois apresentam grande impacto na ciência e indústria. De fato, a computação científica tem se intensificado desde 1990 devido à evolução na capacidade de processamento dos computadores. Enquanto no passado dependia-se muito do poder de processamento de supercomputadores, hoje, conforme (PIRONNEAU, 2000), cerca de 95% dos problemas de computação científica podem ser resolvidos em equipamentos baratos.

Um "ambiente de solução de problemas" (PSE – Problem solving environment) ou simplesmente "software científico" é um ambiente de software integrado que permite a construção de programas, compilação e execução de aplicações para resolução de problemas científicos. Trata-se de um software que oferece ferramentas de alto nível para solucionar um problema, permitindo ao usuário definir estratégias, visualizar e analisar resultados. (RICE & BOISVERT, 1996) refere-se ao Ambiente de Solução de Problemas como sendo um sistema computacional que oferece um conjunto de ferramentas para facilitar a resolução de problemas de uma determinada classe, podendo incluir métodos de solução avançados, ferramenta para seleção de métodos de forma automática ou semi-automática e inclusive oferecer a possibilidade de incorporar novos métodos de solução.

Os softwares científicos podem ser classificados de duas formas distintas:

- sistemas de propósito geral;

- sistemas de domínio específico.

Os sistemas de domínio específico procuram fornecer recursos para atender a um determinado grupo de cientistas, pois tratam de um certo domínio do conhecimento, isto é, suas ferramentas foram construídas para lidar com uma determinada especialidade da ciência. Já os sistemas de propósito geral buscam oferecer um conjunto de recursos amplo e geral, de tal forma que possam ser aplicados em diversos domínios de conhecimento. Alguns sistemas de propósito geral permitem estender as funcionalidades do ambiente através do acoplamento de plug-ins para possibilitar ao cientista o suporte mais apropriado às atividades de seu domínio.

O indivíduo que desenvolve atividades de pesquisa em computação científica é denominado de "cientista computacional". Normalmente esta pessoa é um cientista, um pesquisador, um engenheiro, matemático ou especialista de alguma área do conhecimento, muitas vezes com ampla habilidade em matemática. O ambiente de software científico tem como objetivo aumentar a produtividade do cientista computacional.

2.1 Metodologia dos ambientes de solução de problemas

Em geral, o processo para solucionar um problema é organizado em 7 etapas, como pode ser visto no diagrama da Figura 1.

A primeira etapa consiste em explorar o problema, definindo que questões devem ser respondidas e quais saídas são esperadas. Além disso, deve-se determinar que dados de entrada estão disponíveis para serem utilizados na abordagem. Segundo (ETTER, 2006), esta é a fase mais complexa de todo o processo.

A etapa seguinte (etapa 2) é marcada pela especificação do modelo matemático a ser construído para resolver o problema. Conforme (ETTER, 2006), para criar o modelo matemático é preciso:

- determinar que princípios fundamentais são aplicáveis;
- esboçar o problema para melhor entendimento;
- definir variáveis de entrada e de saída e suas respectivas notações;
- transformar o problema em termos puramente matemáticos;
- simplificar o problema de tal forma que seja o suficiente para obter os resultados desejados;

- identificar e justificar suposições e restrições inerentes ao modelo proposto.

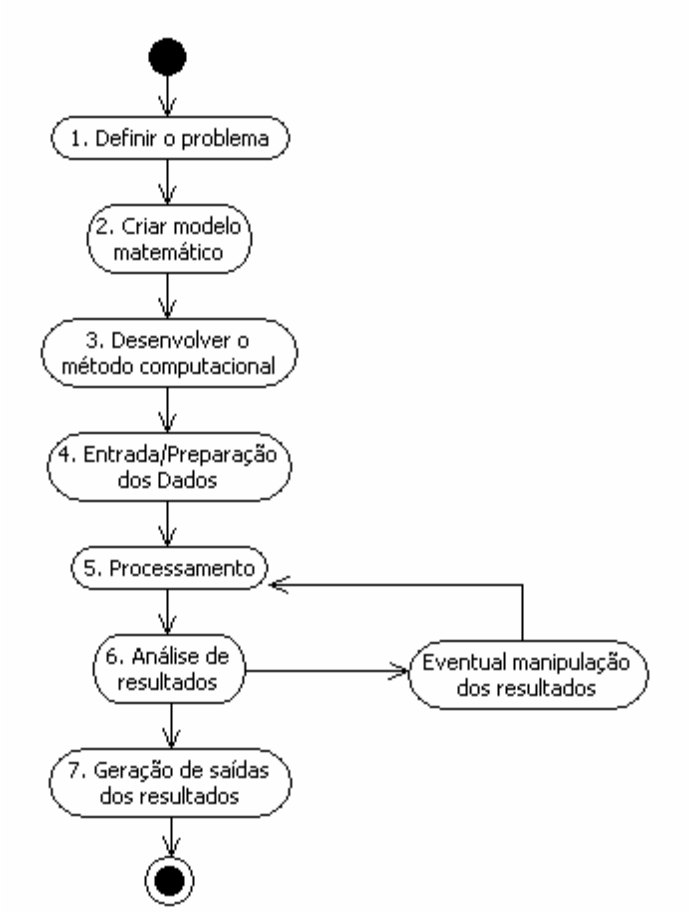

Figura 1 - Fases de um ambiente de solução de problemas

Uma vez que o problema foi descrito em termos matemáticos, a próxima etapa (etapa 3) consiste em determinar os recursos computacionais necessários para construção de um método computacional a ser utilizado para solucionar o problema, além da implementação do método propriamente dito.

Após construir o método computacional, ocorre a entrada dos dados (etapa 4), Esta etapa envolve o levantamento e preparação da entrada de dados (alocação de memória, construção de estruturas de dados especializadas e fornecimento de parâmetros de entrada iniciais).

A próxima fase (fase 5) é caracterizada por uma atividade de intenso processamento realizada a partir de algoritmos complexos do método computacional, sobre os dados de entrada. Esta é a fase de resolução do problema.

Uma característica da computação científica é que ela lida com dados quantitativos (HEATH, 2002). São exemplos de dados quantitativos a distância, velocidade, temperatura e pressão. Assim a atividade de processamento baseia-se essencialmente na aplicação de técnicas de análises numéricas para solucionar os problemas. As técnicas modernas de análise numérica não procuram resultados exatos porque em muitos casos, obter resultados exatos é impossível na prática. Por isso os algoritmos geralmente tentam aproximar-se dos valores reais, isto é, os algoritmos buscam valores aproximadamente corretos ou suficientemente próximos do resultado desejado para propósitos práticos. Há portanto uma fase da aplicação científica (fase 6) que se preocupa em analisar os resultados para verificar se seus valores são satisfatórios. No caso dos resultados não estarem atendendo à necessidade, uma nova iteração do algoritmo deve ser realizada para refinar os resultados. Portanto, a fase de processamento pode ser iterativa. A cada nova iteração espera-se que ocorra uma convergência dos resultados obtidos para os resultados reais. Geralmente, o método processa a primeira iteração introduzindo valores arbitrários ou eventualmente, valores obtidos a partir de algum algoritmo heurístico e a cada nova iteração que é concluída, utiliza-se algum critério de convergência para decidir se os resultados estão próximos do desejável. Um dos aspectos mais importantes da computação científica é encontrar algoritmos que acelerem a convergência dos resultados para os valores reais, além de melhorar a eficiência da aproximação do resultado.

Como a computação científica lida com valores aproximados, há a necessidade de mensurar o grau de precisão da aproximação, isto é, calcular o quão próxima do valor real ela é. Uma medida de erro absoluto é calculada a partir da expressão seguinte:

$$\text{Erro Absoluto} = \text{Estimativa} - \text{Valor Real}$$

Como o valor real geralmente é desconhecido, seu valor precisa ser estimado. Se este erro for tolerável, o software poderá prosseguir com a próxima etapa, caso contrário, uma nova iteração poderá ser executada.

Como etapa final (etapa 7), tem-se a apresentação dos resultados ao usuário que pode ser feita através de forma textual ou a partir de gráficos. Para fins de análise, geralmente a apresentação dos resultados em formato gráfico é mais eficiente. A área da computação que estuda técnicas para criação de representações gráficas dos resultados do processamento de algoritmos científicos é denominada de Visualização Científica. O

objetivo desta área de estudo é auxiliar a compreensão dos resultados pelo cientista e facilitar a aquisição de novos conhecimentos através de interfaces mais informativas.

2.2 Resolução de problemas numéricos

Conforme visto, os problemas computacionais exigem a execução iterativa de algoritmos numéricos, sendo comum a utilização de métodos matemáticos variados. A implementação destes métodos freqüentemente envolve os seguintes conhecimentos:

- operações vetoriais e matriciais;
- funções de interpolação;
- integrações de funções;
- sistemas de equações lineares;
- transformações Fourier discretas;
- equações não lineares e otimização;
- uso de ferramentas estocásticas.

A primeira análise a se fazer sobre um problema matemático é identificar se o mesmo tem uma solução. Quando um problema possui uma única solução e sua solução depender continuamente dos seus dados, isto é, quando ocorrer uma pequena mudança nos dados isso se refletir de maneira proporcional na solução, diz-se que o problema é *well-posed*. Um problema na qual não se pode provar que existe uma solução ou que a solução seja única é denominado *ill-posed*.

Os problemas *well-posed* possuem propriedades importantes para a computação numérica, conforme (HEATH, 2002) descreve, principalmente no que se refere à estabilidade da solução em relação à entrada de dados.

Quanto mais rápida for a convergência para os valores reais, mais eficiente é o algoritmo. Por isso, os pesquisadores estão constantemente procurando desenvolver algoritmos mais otimizados e eficientes. De acordo com (HEATH, 2002), uma abordagem tradicional para minimizar o tempo de convergência é transformar um problema complexo num problema mais fácil, isto é, tratar de um problema difícil usando a solução para um problema similar, porém mais fácil e mais rápido para ser computado. Infelizmente, apesar desta ser a situação ideal, nem sempre é possível obter soluções idênticas. Neste caso é admissível utilizar a solução de um problema alternativo cujo resultado seja relativamente próximo do resultado esperado, isto é,

cujos valores estejam dentro de um determinado limite de tolerância. Alguns exemplos desta abordagem são:

- transformar problemas não lineares em problemas lineares;
- transformar equações diferenciais em equações algébricas;
- transformar espaços de dimensão infinitos em espaços de dimensão finitos;
- transformar sistemas de alta ordem em sistemas de baixa ordem.

(DONGARRA & SULLIVAN, 2000) publicaram uma relação dos 10 melhores algoritmos do século XX, que segundo os autores, representam os algoritmos com maior influência no desenvolvimento e práticas de ciência e engenharia. A lista é compostas por:

- o método Monte Carlo ou Algoritmo Metrópolis;
- o método simplex de programação linear;
- o método de iteração de sub-spaço Krylov;
- a decomposição de matrizes “Householder”;
- o compilador Fortran;
- o algoritmo QR para cálculo de autovalores;
- o algoritmo de ordenação Quicksort;
- a transformação Fast Fourier;
- o algoritmo de detecção de relações de inteiros;
- o algoritmo Multipole.

2.3 Características do software científico

Conforme (HALE, 2006), o objetivo da tecnologia de software científico é explorar os recursos de hardware e permitir que algoritmos complexos possam ser executados em curto espaço de tempo e com baixo custo. Além disso, a solução deve exigir do usuário o mínimo de conhecimento possível em programação de computadores.

O objetivo deste tópico é descrever os aspectos de software desejáveis para a construção de softwares científicos. Um conjunto de características desejáveis para o desenvolvimento de softwares científicos está relacionado a seguir, conforme levantado por (BOISVERT, 2000) e (HOUSTIS, 2003) e são abordados adiante em detalhes.

- abstração de conceitos avançados de engenharia de software;
- reutilização de bibliotecas;
- execução em lote;
- compatibilidade com diversas linguagens de programação;
- desempenho;
- fornecer documentação;
- permitir colaboração com outros cientistas;
- persistência de sessões.

A construção de um software científico geralmente exige o conhecimento de várias disciplinas de ciências da computação. Os arquitetos do software raramente possuem estes conhecimentos, conforme (GALLOPOULOS et al., 1995) relata.

2.3.1 Abstração de conceitos avançados de engenharia de software

O ambiente de software deve fornecer condições para o cientista utilizar recursos de programação sofisticados e processamento matemático otimizado, sem que para isso sejam exigidos do pesquisador conceitos avançados em engenharia de software. O cientista computacional possui um conhecimento profundo sobre a sua ciência de domínio e pode possuir conhecimento avançado em matemática, porém geralmente possui conhecimento limitado sobre programação e outros aspectos de engenharia de software. (RANNA & SHIELDS, 2000) cita que, assim como engenheiros de software não têm intenção de despender anos para se tornar um especialista num determinado domínio, o cientista ou engenheiro também não têm intenção em tornar-se um especialista em ciências da computação. O ambiente computacional deve portanto fornecer uma estrutura intuitiva e fácil para o cientista computacional. O software deve permitir que o especialista se concentre na sua disciplina, assim as informações ou conhecimentos de ciências da computação precisam ser abstraídos sempre que possível.

O ambiente de software deve fornecer uma interface que utilize uma linguagem apropriada para o domínio do problema, dessa forma, os cientistas podem interagir com o sistema sem ter conhecimento especializado de hardware e software, abstraindo a plataforma de sistema operacional, protocolos de rede e linguagem de programação.

Este objetivo pode ser atingido utilizando-se algumas tecnologias:

- utilização de interface gráfica a fim de tornar fácil a aprendizagem do software;
- utilização de um módulo para tomada de decisão, que associado a uma base de conhecimento, auxilia o cientista a escolher a melhor técnica para solucionar um determinado problema.

De acordo com (ABRAMS, 2006), muitos problemas e estratégias de solução são extremamente heterogêneas: em modelos, aplicações, máquinas e até em terminologia. Um aspecto desafiador de um bom software científico é gerenciar estes componentes heterogêneos de uma forma integrada, para que pelo menos o usuário tenha a impressão de estar trabalhando num único ambiente de software consistente.

Em domínios de problemas complexos, o software pode fazer uso de técnicas de inteligência artificial para interagir com o usuário e sugerir métodos (isto é, algoritmos, componentes de software, recursos de hardware...) mais apropriados para resolver um determinado problema (GOEL et al., 1999).

2.3.2 Reutilização de bibliotecas

O software científico deveria permitir a reutilização de bibliotecas (módulos) de software desenvolvidas por terceiros para evitar que o cientista tenha que se preocupar com a construção de técnicas já construídas no passado. Algumas técnicas são freqüentemente utilizadas pelos pesquisadores e o software científico deveria permitir que o cientista aproveitasse tais técnicas sem que fosse necessário reescrevê-las.

Um ambiente de software científico deveria suportar o paradigma “plug-and-play”, de tal forma que seja possível incorporar dinamicamente novos componentes de software. Conforme (GALLOPOULOS et al., 1997), isto normalmente envolve a criação de um framework com interfaces definidas formalmente. Alguns componentes importantes são sugeridos como, por exemplo, manipulação de modelagem geométrica, visualização científica, matemática simbólica, bancos de dados, etc.

Existem diversas bibliotecas de técnicas reutilizáveis disponíveis gratuitamente, tais como LAPACK (Linear Álgebra Package), PETSc (Portable, Extensible Toolkit for Scientific Computation) e ARPACK (Arnoldi Package) (HYARIC, 2007). Apesar desta característica ser importante, o software científico precisa abstrair a complexidade na

utilização das bibliotecas para evitar que a curva de aprendizagem do cientista iniba ou dificulte a sua utilização.

A linguagem R, utilizada neste trabalho, destaca-se por possuir diversas técnicas estatísticas já implementadas, como por exemplo: modelagem linear e não linear, testes estatísticos clássicos, análises de séries temporais, classificação e agrupamento (*clustering)* (VENABLES & SMITH, 2006). Seus principais recursos envolvem a manipulação de dados e geração de gráficos para análise de dados.

2.3.3 Execução em lote

Apesar de se esperar que o software científico disponibilize interfaces gráficas, ainda assim, é desejável poder acessar os recursos do software através de linguagem de script ou linha de comando. O motivo disso é que alguns algoritmos podem consumir muito tempo para serem executados, levando inclusive alguns dias para serem concluídos. Neste caso, o cientista não irá acompanhar a execução do algoritmo e portanto, não poderá operar a interface gráfica. Uma alternativa é oferecer ao cientista uma linguagem de script para operar o software e pré-programar sua execução. Este tipo de execução é denominado de execução em lote, pois um lote de comandos é submetido inicialmente e uma única vez ao software e este executa os comandos um após o outro, sem que seja feita requisição para o usuário.

2.3.4 Compatibilidade com diversas linguagens de programação

Uma característica desejável num ambiente de software científico é a de permitir a execução de programas escritos em diversas linguagens. Há duas razões para isto. Primeiro, como existem muitas técnicas já desenvolvidas em diversas linguagens, torna-se extremamente útil aproveitar estes trabalhos. Segundo, ao oferecer para o cientista uma linguagem de programação já conhecida, diminui-se o esforço necessário para que o cientista adapte-se à nova linguagem, facilitando inclusive aos programadores mais conservadores a adoção ao software.

Segundo (HALE, 2006), a linguagem de programação pioneira para programação científica é a linguagem Fortran, por sua eficiência e facilidade em expressar fórmulas matemáticas. Como foi a primeira linguagem a ser utilizada para programação científica, muito já foi produzido e investido com esta linguagem e por isso é desejável que o software científico seja compatível com esta linguagem.

No caso da linguagem Fortran, a abordagem de programação tradicional envolve a utilização de bibliotecas e programação estruturada.

Nos dias de hoje, Fortran já não atrai a mesma atenção que antigamente. (LUJÁN, 2000) cita que Fortran é uma linguagem de programação ultrapassada e tem sido utilizada com pouca freqüência pelos desenvolvedores de software, que têm dado preferências às linguagens mais populares e modernas como C, C++ e Java.

De acordo com (ARNOLD & DONGARRA, 2000), a linguagem C não foi planejada para ser utilizada para construir aplicações científicas. Ao contrário de outras linguagens, C é uma linguagem próxima da linguagem de baixo nível e por isso permite acesso direto às instruções do computador. Além disso, C possui diversos tipos de dados e permite aritmética com ponteiros. Estes recursos apesar de oportunizar a construção de aplicações sofisticadas, exigem maior conhecimento de arquitetura de hardware e software. Porém, devido às características de universalidade, portabilidade e flexibilidade, a linguagem C tem atraído um número significativo de engenheiros e cientistas.

Outras linguagens também estão sendo utilizadas, como Eiffel, Octave, Num-Python, Sci-Python e PDL.

Algumas características desejáveis do ambiente de software são, conforme (HALE, 2006):

- linguagem gratuita;
- suportar vários sistemas operacionais;
- suportar bibliotecas de código livre escritas em outras linguagens;
- portabilidade.

Vale a pena mencionar que, apesar de ser desejável manter compatibilidade com diversas linguagens, esta característica pode elevar muito o custo do projeto do software.

2.3.5 Desempenho

Devido aos requisitos de capacidade de processamento, espera-se que o software científico tenha bom desempenho e isso significa economia e otimização de recursos, como memória e CPU. Muitas vezes, aplicações científicas lidam com conjuntos de dados com centenas de gigabytes e os algoritmos podem consumir inclusive vários dias

para processá-los. A construção de um software científico deve levar em conta os recursos que serão exigidos e utilizar de maneira inteligente os componentes do computador.

Recentemente, tem-se utilizado computação distribuída para permitir processamento paralelo de algoritmos demorados em diversos computadores, aproveitando desta forma, a capacidade de processamento de vários equipamentos. Neste tipo de solução, o ambiente de software deve fornecer ferramentas para automaticamente selecionar um equipamento apropriado entre um conjunto de equipamentos disponíveis, além de gerenciar a distribuição de tarefas dentro da rede de computadores selecionados.

2.3.6 Fornecer documentação

O software científico deveria favorecer a criação de documentação dos algoritmos construídos. Um algoritmo bem documentado facilita a sua utilização por outros usuários e torna mais fácil a sua modificação. Conforme (O'LEARY, 2005), a documentação deveria incluir as seguintes informações:

- propósito do código de programa: trata-se de uma breve descrição da finalidade do programa. Tal informação é útil quando algum cientista estiver à procura de algum programa para ser reutilizado. Neste caso, o propósito do código de programa será necessário para que o cientista se certifique que o programa localizado atenderá sua necessidade. Sem a documentação do propósito do programa, o cientista precisará despender tempo investigando o código de programa a fim de compreender seu funcionamento;
- nome do autor ou mantenedor: para que se saiba para quem se deve reportar possíveis defeitos (*bugs*) encontrados nos programas ou dúvidas a serem esclarecidas;
- data do programa original e lista das modificações: para que o cientista saiba se é necessário aplicar alguma modificação no ambiente computacional para fazer uso de novas versões do algoritmo;
- descrição de cada parâmetro de entrada: consiste numa relação dos parâmetros que devem ser fornecidos ao algoritmo de forma a se garantir o

seu correto funcionamento. Quando aplicável, deve-se descrever também os respectivos tipos de dados ou restrições quanto aos possíveis valores a serem enviados ao algoritmo;

- descrição de cada parâmetro de saída: para que o cientista saiba que informações serão disponibilizadas pelo software logo após a execução do algoritmo;
- descrição do método e referências: para que o cientista possa decidir se o algoritmo é útil para sua necessidade.

Outros detalhes adicionais também são bem vindos como, por exemplo, relação de bugs conhecidos.

2.3.7 Colaboração com outros cientistas

O software deve permitir que diversos cientistas cooperem com o desenvolvimento dos algoritmos. Conforme cita (GALLOPOULOS et al., 1997), a maioria dos projetos de sistemas científicos exige uma equipe de cientistas multidisciplinar, envolvendo pesquisadores distribuídos geograficamente.

Nos dias atuais, graças à internet, muitas equipes de desenvolvimento são constituídas de pessoas espalhadas ao redor do planeta e o software deveria suportar o compartilhamento dos trabalhos dos diversos cientistas.

É desejável também que os recursos computacionais como bibliotecas de software e outros algoritmos estejam disponíveis remotamente para um único cientista.

2.3.8 Persistência de sessões

Um ambiente de software científico deveria permitir o conceito de seções, isto é, o cientista poderia armazenar o seu trabalho num dispositivo de armazenamento secundário para futuramente dar continuidade a partir da seção armazenada. Este recurso se deve principalmente porque para resolver um problema complexo pode ser necessária a dedicação de um tempo significativo. Algumas vezes, contextos parciais de análises poderiam ser armazenados para se estudar possíveis estratégias para etapas posteriores, sem que para isso, seja preciso processar novamente as etapas anteriores.

2.4 Organização de um software científico

Em geral, um software científico está organizado em três camadas, conforme sugerido por (HEY et al., 2002):

- interface do usuário: é a interface utilizada para o usuário interagir com o software, criando e modificando aplicações científicas. Normalmente é uma interface em modo gráfico (GUI – graphical user interface), porém interface em modo texto também pode ser utilizada para execução de comandos em lote. A interface do usuário também pode dispor de modelos para orientar o cientista em compor a solução de novos problemas;
- serviço de *middleware*: a fim de criar a ilusão para o usuário de que o mesmo está operando uma plataforma de hardware e software homogênea, mesmo que o ambiente integre tecnologias distintas, faz-se necessário utilizar uma arquitetura que forneça este serviço. A arquitetura SOA permite criar esta abstração. Ainda na camada intermediária, deve existir um serviço de documentação para permitir a documentação de suas aplicações e módulos;
- bibliotecas de software, componentes/pacotes: é a camada mais interna e é composta pelas funções para processamento.

2.5 Exemplos de softwares científicos

Existem no mercado alguns softwares disponíveis para análise de problemas científicos. Alguns sistemas são ditos de "propósito geral", isto é, são sistemas que não se limitam a um determinado domínio de estudo. Pode-se citar nesta categoria: MATLAB (Mathworks, 2007), Macsyma (SYMBOLICS, 1995), Mathematica (WOLFRAM, 2006), Maple (Maplesoft, 2007) e SOC (BRASIL, 2007). Outros sistemas foram contemplados para atender problemas específicos. Uma relação ampla de softwares científicos pode ser encontrada em http://www.cs.purdue.edu/research/cse/pses/research.html.

2.5.1 Matlab

Matlab é um software para resolução de problemas científicos composto por um ambiente interativo, bibliotecas para computação numérica, geração de gráficos

avançados e uma linguagem de programação de alto nível. Seu desenvolvimento iniciou na década de 70 por Cleve Moler e desde 1984 vêm sendo mantido pela empresa The MathWorks. O objetivo inicial do projeto Matlab era construir um ambiente que permitisse a utilização interativa dos pacotes EISPACK e LINPACK do Fortran para processamento de álgebra linear. O nome Matlab significa Matrix Laboratory, porque o sistema foi projetado para tornar fácil o processamento de matrizes.

As principais características do Matlab são, de acordo com (TREFETHEN et al., 1996) e (KUNCICKY, 2003):

- possui linguagem interpretada para execução de algoritmos. A linguagem Matlab possui todas as características das linguagens tradicionais, permitindo inclusive a programação orientada à objetos;
- possui interface amigável. Além disso, é possível que o próprio usuário desenvolva as interfaces dos seus programas;
- possui sintaxe intuitiva que favorecem a construção de algoritmos utilizando um número menor de instruções e ainda assim mantendo a legibilidade do programa. Por exemplo: enquanto que em C ou Fortran pode ser necessário escrever algumas dezenas ou centenas de instruções para realizar uma determinada computação, em Matlab pode ser suficiente escrever uma ou duas linhas de comandos;
- dispõe de alta qualidade dos programas numéricos desenvolvidos. Mesmo utilizando uma linguagem interpretada possui ótima performance;
- possui ferramentas gráficas sofisticadas e poderosas, com gráficos 2D e 3D;
- facilidade de estender seus recursos através do uso de pacote ou *toolboxes*. Atualmente existe uma grande quantidade de pacotes fornecidos pela própria MathWorks, porém terceiros também podem construir seus próprios pacotes;
- permite a integração de seus algoritmos com aplicativos externos escritos em outras linguagens;
- permite gerar aplicativo em linguagem de máquina a partir do programa construído.

Atualmente o núcleo do Matlab está escrito em C e Java e por isso está disponível em várias plataformas, como Windows, Linux, HP-UX, Mac OS X e Solaris.

Matlab apesar de ser um produto comercial, possui arquitetura aberta, o que facilita o desenvolvimento de novos pacotes. Atualmente existem mais de 40 pacotes desenvolvidos pela própria MathWorks, além de diversos outros pacotes criados por terceiros, entre eles pacotes gratuitos. Alguns dos pacotes mais conhecidos são: bioinformática, estatística, redes neurais, processamento de imagens, processamento de sinais e algoritmos genéticos.

A principal desvantagem do Matlab é o fato de ser um produto comercial e possuir um preço caro, especialmente para ser adquirido por estudantes.

2.5.2 R

R é um ambiente de software científico, dotado de uma linguagem de programação e ambiente para análise de dados e gráficos. O fato do ambiente R possuir uma série de técnicas estatísticas desenvolvidas é uma das razões que o tornaram, conforme (CHAMBERS, 1998), um padrão entre os estatísticos para o desenvolvimento de software estatístico.

O ambiente R foi idealizado em 1995 por Ross Ihaka e Robert Gentleman, ambos da Universidade de Auckland. Atualmente o ambiente R é mantido por um grupo de colaboradores voluntários. O ambiente R está disponível sem nenhum custo e seu código fonte está disponível para acesso através da licença GNU, sendo portanto de livre distribuição. O ambiente R foi inspirado num outro ambiente, não gratuito, denominado S. Muitas funcionalidades do ambiente S estão presentes no ambiente R.

O ambiente R pode ser executado em diversas plataformas de sistemas operacionais, incluindo Windows, plataformas Unix (Linux, FreeBSD) e MacOS.

Existe uma série de técnicas estatísticas desenvolvidas e disponibilizadas para o ambiente R, entre as quais, destacam-se: testes estatísticos clássicos, modelagem linear e não linear, análise de série temporais, classificação e agrupamento. Além disso, R disponibiliza também uma diversidade de funções para construção de gráficos.

O ambiente R dispõe de uma linguagem de programação denominada R. A linguagem R é uma linguagem de alto nível com possibilidade de manipular os dados utilizando-se operações mais intuitivas que as utilizadas pelas linguagens tradicionais.

A linguagem R é uma linguagem funcional, pois dá ênfase à aplicação de funções e avaliação de expressões, ao contrário das linguagens de programação tradicionais – denominadas imperativas – na qual se dá ênfase às mudanças de estado e à execução de comandos seqüenciais. Por isso, em R não há alocação de memória explícita nem declaração de variável explícita. Estas operações são invocadas dinamicamente pela linguagem sempre que for necessário. R também não utiliza arquivos de cabeçalho (headers) e não há ponteiros para memória. A construção da linguagem R utiliza um modelo de computação baseado no dialeto Scheme da linguagem LISP (MAINDONALD & BRAUN, 2003).

A sintaxe da linguagem R é similar à linguagem C, porém a linguagem R possui muitos outros recursos desenvolvidos para simplificar a manipulação de dados numéricos. Por exemplo: considerar uma variável do tipo vetor chamada "dados" com 5 elementos. Em C, para criar um novo vetor contendo a raiz quadrada de cada componente do vetor "dados", utiliza-se um programa similar ao visto no Quadro 1:

```
int raiz(5);
for (c=1; c<=5; c++)
  raiz(c) = srqt(dados(c));
```

Quadro 1 - Exemplo de código na linguagem C

Já na linguagem R, a mesma operação pode ser feita com uma única instrução, de maneira mais simples e elegante, conforme pode ser visto no Quadro 2. Esta abordagem evita a utilização de laços explícitos tornando o código de programa mais claro.

```
raiz = srqt(dados);
```

Quadro 2 - Exemplo de código na linguagem R

Um outro recurso do software R é a capacidade de estender as funcionalidades do ambiente através do paradigma de pacotes (bibliotecas). As bibliotecas podem ser instaladas dinamicamente e podem ser baixadas da internet utilizando-se o próprio ambiente. Muitas bibliotecas foram construídas na própria linguagem R, mas é possível utilizar as linguagens C, C++ e inclusive a linguagem Fortran caso seja imperativo obter um ótimo desempenho.

A linguagem R é orientada a objetos, porém por ser uma linguagem funcional, utiliza uma abordagem diferente da utilizada nas linguagens tradicionais como C++ e Java: ao invés dos objetos possuírem métodos, as funções é que possuem métodos que

se comportam conforme o tipo de objeto que está sendo tratado. Existe em R duas técnicas para a construção de classes e instanciação de objetos, denominadas de “S style” e “New Style”.

Algumas outras características do ambiente R são:

- possui um bom sistema de ajuda;
- possui recursos avançados para geração de gráficos;
- a própria linguagem é extensível.

Tanto R como Matlab são softwares populares entre a comunidade científica e são exemplos de como a tecnologia contribui para o desenvolvimento da ciência. Portanto, a pesquisa e desenvolvimento de ambientes de software que promovam a solução de problemas científicos são essenciais para o crescimento da ciência.

3 APLICAÇÕES BASEADAS EM PLUG-INS

É inevitável que qualquer software sofra modificações durante seu ciclo de vida. Estas modificações podem ser com intuito de evoluir o software, isto é, incluir novas funcionalidades ao produto, ou para corrigir falhas encontradas no software. A forma mais tradicional de realizar estas modificações é, segundo (CHATLEY, 2005), recompilando o código fonte do software e substituindo a versão antiga, pela nova versão recompilada. Entretanto, isto introduz alguns problemas, como o surgimento de novos defeitos, a necessidade de interromper a execução do aplicativo atual para substituí-lo pela nova versão, etc. Uma estratégia para evitar estes problemas é desenvolver o software utilizando o conceito de plug-ins.

3.1 Plug-ins

O termo plug-in é utilizado para representar um tipo de programa que é utilizado para agregar novas funções a outros programas. Para que um plug-in seja utilizado por outra aplicação, esta deve suportar o mecanismo de plug-in. Denomina-se aplicação baseada em plug-ins todo software capaz de incorporar novas funcionalidades ou novas características a partir do acoplamento de plug-ins. A Figura 2 demonstra uma aplicação sendo estendida por três diferentes plug-ins.

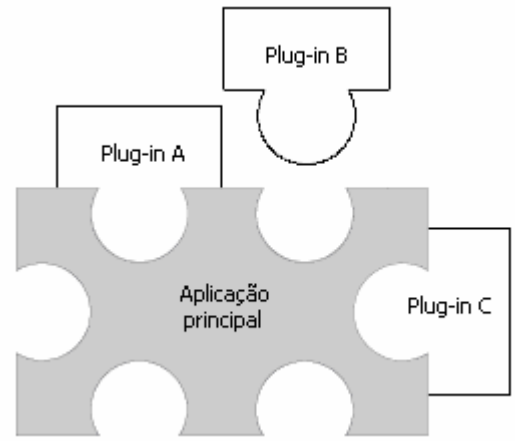

Figura 2 - Estendendo a aplicação através de plug-ins

Para um plug-in ser incorporado a uma aplicação, o plug-in deve conhecer a interface de comunicação da aplicação. Na Figura 2, as concavidades representam uma

interface de comunicação exposta pela aplicação. O plug-in somente pode ser integrado à aplicação principal se o plug-in atender à interface de comunicação.

Um exemplo típico de aplicação que é capaz de aproveitar o mecanismo de plug-ins é a aplicação que manipula arquivos multimídia, como por exemplo, um software para visualizar arquivos de imagens. O desenvolvedor deste tipo de aplicação pode não ser capaz de construir o mecanismo para processar todos os tipos de arquivos multimídias que existem e os futuros tipos de arquivos que eventualmente serão construídos. Este problema pode ser resolvido ao permitir que o processamento de imagens possa ser desenvolvido através de plug-ins. Desta forma estas características adicionais podem ser fornecidas pelo software sem que seja necessário atualizar o aplicativo completamente, sendo apenas necessário instalar o novo plug-in para tratar do novo tipo de arquivo multimídia.

Alguns exemplos de aplicações que fazem uso do conceito de plug-ins estão relacionados a seguir (MAYER et al., 2002) e (CHATLEY, 2005):

- navegador de internet baseados na tecnologia Gecko, tal como o software Firefox. Cada plug-in é especializado em tratar algum tipo de arquivo. A maior parte dos plug-ins processa arquivos multimídia. Os populares arquivos de animação Shockwave Flash, da Macromedia, são acessíveis a partir de navegadores internet, pois o fabricante desenvolveu plug-ins para reprodução dos seus arquivos;
- o ambiente de desenvolvimento de programas Eclipse é altamente extensível por plug-ins. Quase todas as funcionalidades do ambiente estão disponíveis através de plug-ins. O ambiente suporta o desenvolvimento em diversas linguagens de programação, utilizando um plug-in para cada linguagem. O ambiente Eclipse permite inclusive que plug-ins possam estender a funcionalidade de outros plug-ins;
- a ferramenta de processamento de imagens ImageJ utiliza dois tipos de plug-ins: plug-ins que lidam com entrada de arquivos e outros que atuam como filtro de imagens.

Um plug-in não é conhecido durante a compilação da aplicação, isto é, não há qualquer referência a um determinado plug-in no código fonte da aplicação. Devido a esta característica, outros desenvolvedores (e não somente os autores da aplicação),

podem contribuir com a construção de novas funcionalidades para uma aplicação baseada em plug-ins.

Há várias razões do porquê construir aplicações baseadas em plug-ins (CHATLEY, 2005):

- a necessidade de adicionar novos recursos ao software. O mecanismo de plug-ins é utilizado justamente para permitir a adição de recursos a uma aplicação já existente. O fabricante ou autor do software pode permitir que desenvolvedores terceiros possam construir plug-ins para sua aplicação, desde que para isso seja fornecida a especificação da interface de como escrever plug-ins para a aplicação;
- a decomposição de grandes sistemas em módulos menores. A manutenção de software é uma preocupação constante do processo de desenvolvimento de software e uma das estratégias para minimizar os esforços em se lidar com sistemas complexos é dividir um sistema em pequenas partes (módulos). A organização de software em módulos permite que o desenvolvimento fique restrito a construção daquela funcionalidade;
- atualização de software sem que o sistema precise ser reinicializado. Em sistemas que requerem alta disponibilidade pode ser inconveniente interromper a execução do software para atualizá-lo. Com a tecnologia de plug-ins, pode-se adicionar um plug-in sem que a aplicação seja reinicializada. Desta forma, a atualização de um plug-in requer apenas interromper a execução do plug-in a ser atualizado. O restante da aplicação permanece em execução, sem ser comprometida;
- economia de recursos do computador: Numa aplicação baseada em plug-ins, a alocação de memória para os plug-ins ocorre sob demanda, isto é, assim que requisitado o plug-in, o mesmo é carregado para a memória do computador. O mecanismo de plug-ins permite que somente seja alocada memória para os recursos utilizados. Mesmo que o software seja rico em recursos, somente os recursos utilizados num determinado momento é que ocuparão memória. Como nem todos os plug-ins devem ser solicitados simultaneamente pelo usuário, o consumo médio de memória será inferior que o consumo de uma aplicação tradicional.

Além disso, de acordo com (MAYER et al., 2002), como normalmente uma aplicação sofre evoluções constantes, freqüentemente os programadores não sabem quais partes antigas estão em uso (pelos usuários) exigindo que tais partes antigas não possam ser removidas. Com o uso de plug-ins, novos recursos podem ser construídos sem comprometer as funcionalidades oferecidas por outros plug-ins.

3.2 Padrão de projeto para o mecanismo de plug-ins

Um padrão de projeto é um modelo utilizado para resolver um tipo de problema específico. Conforme (GAMMA et al., 1995), padrões de projeto são soluções elegantes e reutilizáveis para problemas recorrentes que são encontrados no processo de desenvolvimento de aplicativos. A adoção de padrões de projeto torna o software mais fácil de entender, simplificando o processo de manutenção de sistema. Além disso, os padrões definem um vocabulário comum contribuindo para uma melhor comunicação entre os participantes do projeto de desenvolvimento do software.

Um padrão de projeto para a construção de mecanismos de plug-in é apresentado por (MAYER et al., 2002). Este padrão de projeto explica como elaborar uma aplicação para suportar o conceito de plug-ins a fim de estender em tempo de execução suas funcionalidades através do carregamento dinâmico de módulos ou classes não conhecidas durante a compilação da aplicação. O diagrama UML sugerido pelo autor, para um padrão de projeto "plug-in" pode ser visto na Figura 3.

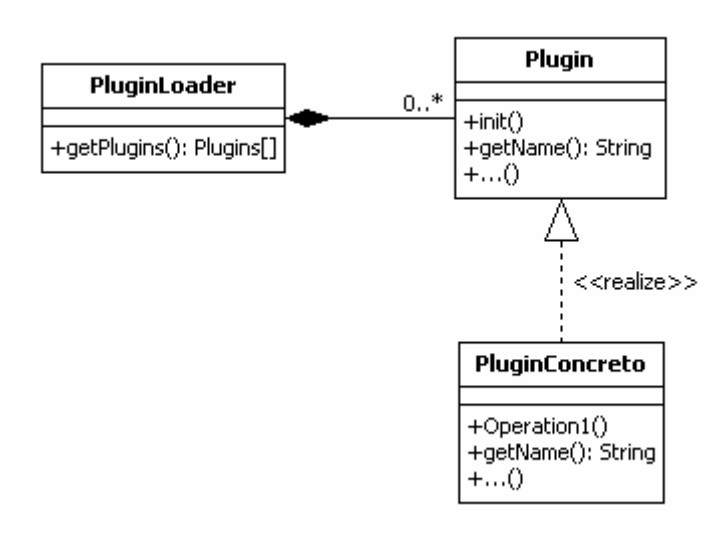

Figura 3 - Padrão de projeto "plug-in"

Neste diagrama, a classe Plugin não representa nenhuma implementação de plug-in em específico. Ao invés disso, Plugin é uma interface que descreve como os plug-ins devem se construídos, sendo portanto, a classe pai para os plug-ins desenvolvidos. Fazendo a analogia com a Figura 2, as concavidades representam esta interface.

O método *getName( )* de Plugin é utilizado para que o plug-in possa informar qual é o seu nome, útil por exemplo, caso o plug-in precisar informar o seu nome para ser exibido como item de menu. O autor também sugere que a classe Plugin possa conter métodos a serem invocados pela aplicação para determinar se o plug-in deve ser executado em determinado contexto ou não.

Enquanto a classe Plugin é uma especificação abstrata, a classe PluginConcreto trata de uma implementação real de um plug-in. Um objeto PluginConcreto deve necessariamente implementar todos os métodos da sua interface (Plug-in).

A classe PluginLoader é responsável por instanciar os plug-ins. É o PluginLoader que carregará um plug-in para a memória, chamando para isso, seu método *init*( ). Esta classe é compilada com a aplicação.

Para a implementação do padrão de projeto Plug-in a determinação de sub-classes para uma certa interface deve ser realizada em tempo de execução. Entretanto, não existe linguagem de programação que suporte este comportamento (MAYER et al., 2002). Uma abordagem freqüentemente utilizada é fazer uma pesquisa no sistema de arquivos em busca dos arquivos correspondentes aos plug-ins. Neste caso, PluginLoader é a classe responsável por realizar esta tarefa.

A estrutura de uma aplicação baseada em plug-ins, segundo este padrão de projeto, está ilustrada na Figura 4:

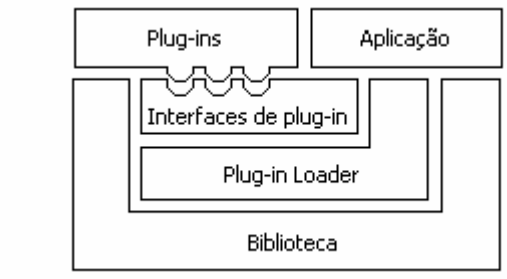

Figura 4 - Estrutura de uma aplicação baseada em plug-ins

A camada Interfaces de plug-in corresponde a uma ou mais interfaces de plug-in, permitindo assim, vários tipos ou categorias de plug-ins. Um plug-in necessariamente

deve satisfazer uma interface, porém é possível que um único plug-in seja compatível com várias interfaces.

Os plug-ins são instanciados em tempo de execução da aplicação pela camada Plug-in Loader. É possível que exista uma camada "plug-in loader" para cada tipo de plug-in. Os plug-ins são acessados a partir do Plug-in Loader ou através de suas interfaces.

A camada "Biblioteca" contém funções gerais que podem ser reutilizadas por qualquer plug-in e inclusive a própria aplicação principal.

Uma outra abordagem para desenvolvimento de sistemas baseados em plug-ins é proposta por (VÖLTER, 1999). O diagrama de classe é apresentado na Figura 5.

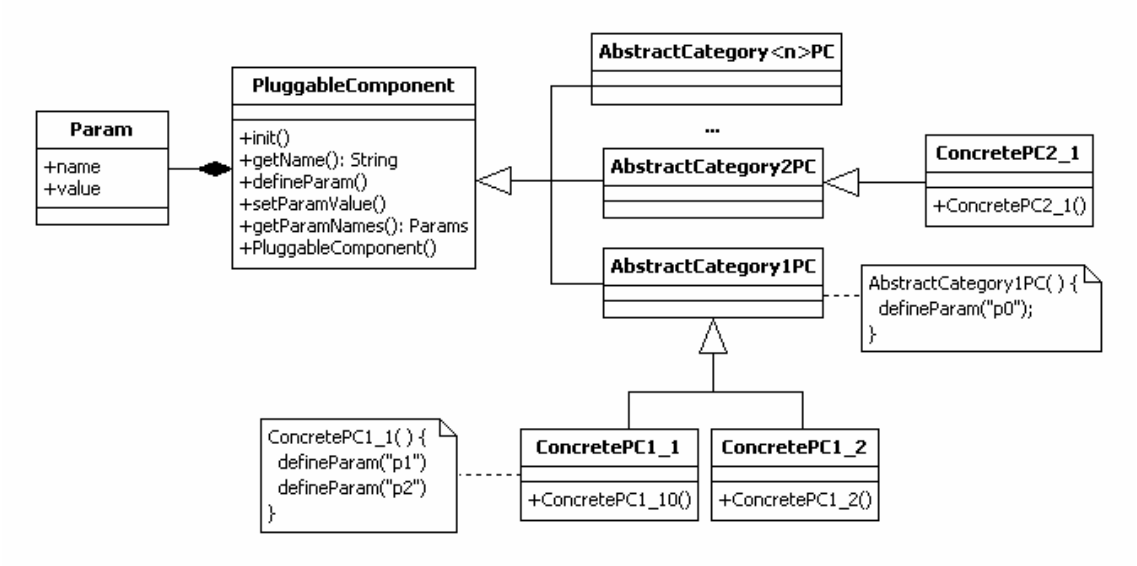

Figura 5 - Proposta de padrão de projeto para plug-ins

Este padrão de projeto possui similaridades com o padrão apresentado anteriormente, como o uso da classe PluggableComponent que define o comportamento e operações padrões para qualquer plug-in e a classe ConcretePC2_1 que representa um plug-in concreto. Neste diagrama estão explicitadas também as diversas categorias de plug-in permitidas pelo software, na qual pode ser visto através das interfaces AbstractCategory<n>PC.

A classe PluggableComponent possui o método *init*( ) que deve ser utilizado para iniciar um plug-in e o método *getName*( ), que deve ser utilizado para obter o nome do componente. Além disso, PluggableComponent é responsável em manipular uma coleção de parâmetros úteis para que um determinado plug-in possa ser utilizado. Estes parâmetros são utilizados para armazenar as informações de configuração necessárias

para o plug-in se comunicar com a aplicação. O método *defineParam*( ) é utilizado para declarar um novo parâmetro, enquanto que o método *setParamValue*(name, value) é utilizado para atribuir o valor value para o parâmetro name.

A classe Param é caracterizada pelo atributo name que representa o nome do parâmetro, enquanto que o atributo value armazena o valor do parâmetro. Uma implementação robusta desta estrutura deve utilizar um tipo de dado polimórfico. Em Java, por exemplo, pode ser utilizado o tipo de dado Object.

Este padrão de projeto permite a construção de categorias de plug-in. Cada categoria deve necessariamente possuir uma interface, como em AbstracCategory1PC. Uma categoria de plug-in pode possuir seus próprios parâmetros. Novas categorias devem derivar de PluggableComponent.

Qualquer plug-in concreto precisa se enquadrar numa categoria previamente definida, por isso o plug-in deve atender alguma interface, conforme sua categoria. É permitido ao plug-in definir novos parâmetros, conforme necessário.

3.3 Tecnologias para desenvolvimento de plug-ins

A idéia de construir um software em partes ou componentes, de tal forma que estes componentes possam ser criados por diferentes desenvolvedores, usando diferentes linguagens e diferentes plataformas é um paradigma proposto em 1969 por McIlorys, denominado de Desenvolvimento de Software baseado em Componentes (DSBC). A DSBC é considerada atualmente como uma das propostas mais promissoras para desenvolvimento de software (WONG et al., 2000). O objetivo principal do DSBC é a redução do custo de desenvolvimento de software, que é alcançada devido à reutilização no processo de desenvolvimento, além disso, o processo de manutenção de sistemas baseados em componentes apresenta flexibilidade e certa facilidade (MAHMOUD, 1999).

Para (MAYER et al., 2002), um componente é um elemento de software que satisfaz as seguintes condições:

- pode ser utilizado por outros elementos de software, denominados seus "clientes";
- possui uma descrição de uso, que é suficiente para que o desenvolver de um cliente possa utilizá-lo;

- não está limitado a nenhum conjunto de clientes.

Embora a forma mais comum de se reutilizar componente seja a nível de implementação, outros artefatos também podem ser reutilizados. De acordo com (HERZUM, 1999), todos os aspectos e fases do ciclo de vida do desenvolvimento de um software, incluindo análise de requisitos, arquitetura, projeto, construção, testes, distribuição, suporte técnico, e a gerência de projeto, são baseados em componentes.

Em geral, os componentes de software são constituídos por nome, interface e código, conforme apresentado na Figura 6:

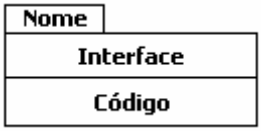

Figura 6 - Composição geral de um componente

Todos os serviços fornecidos pelo componente estão implementados através do Código do componente. O código não é visível ou acessível pelo mundo exterior. A única forma de acessar os serviços do componente é através da Interface. A interface apresenta a forma como o mundo exterior, ou o cliente, segundo (MAYER et al., 2002), acessa o componente, para solicitar seus serviços.

A forma como componentes são desenvolvidos é determinada pelo modelo de componentes adotado. Conforme (LAU & WANG, 2007), um modelo de componentes define a arquitetura básica de um componente, especificando a estrutura de sua interface e os mecanismos pelo qual suas instâncias interagem com outras instâncias de componentes e com o mundo externo. O modelo de componentes fornece uma orientação sobre como criar e implementar um componente para comunicar com outros componentes a fim de criar uma aplicação.

(DEPRINCE JR & HOFMEISTER, 2002) citam que os modelos de componentes evoluíram muito e continuam a evoluir. Ao optar por um modelo de componentes se está determinando que linguagens de programação e plataformas de hardware e software serão suportadas pelo componente.

Este tópico deste capítulo apresenta alguns modelos de componentes que podem ser utilizados para desenvolver aplicações baseados em plug-ins, que, conforme (CHATLEY, 2005), se encaixam perfeitamente na descrição de componentes.

3.3.1 Javabeans

Javabeans é um modelo de componente reutilizável escrito em linguagem Java e proposto pela Sun em 1997. Este modelo descreve como os componentes são manipulados e como eles interagem com o mundo externo, além de ser responsável por uma variedade de serviços (MORRISON, 1997).

O objetivo principal de Javabeans é a de permitir a construção de aplicativos utilizando componentes desenvolvidos por terceiros e que possam ser manipulados através de uma ferramenta visual, isto é, Javabeans foi projetado para ser utilizado sobretudo em interfaces gráficas.

Diferente de outros modelos de componentes, a especificação de Javabeans exige que o componente interaja em dois contextos diferentes: um em tempo de desenho (*design time*) e outro em tempo de execução (*run-time*).

A interface exposta pelo Javabeans é composta por:

- métodos;
- propriedades;
- eventos;
- listeners.

Os *métodos* são as operações, serviços ou ações que o componente é capaz de realizar. Em tempo de execução o cliente faz chamadas aos métodos do componente para solicitar a execução de alguma ação.

As *propriedades* são utilizadas para configurar um componente em tempo de desenho ou de execução. Um cliente pode ler propriedades de um componente ou definir suas propriedades.

Os Javabeans contam com o recurso chamado de Introspecção que permite uma aplicação solicitar a um componente qualquer quais são suas propriedades e métodos e mediante estas informações, a aplicação pode interagir com o componente da maneira mais apropriada.

A alteração de propriedades de um Javabean pode gerar um *evento*. Os Javabeans possuem um mecanismo para tratamento de eventos, que permite ao componente notificar a ocorrência de uma mudança de estado. Por exemplo, o clique de um componente visual pode provocar a notificação de que o componente foi submetido a esta ação do mouse.

Os componentes do tipo Javabean também contam com o recurso de persistência, que é utilizado para guardar em meio de armazenamento secundário (não volátil) as propriedades e estados internos de uma instância de componente, de tal forma que possam ser recuperadas mais tarde para restabelecer seu estado anterior.

A implementação de um Javabean é similar a implementação de uma classe em Java, porém deve atender algumas exigências da especificação Javabeans. A especificação Javabeans pode ser encontrada em http://java.sun.com/beans.

O uso de Javabeans permite a construção de aplicações que suportam a inclusão dinâmica de novos componentes de software, já que classes Java podem ser compiladas e distribuídas separadamente, como arquivos individuais. A máquina virtual Java carrega as classes dinamicamente conforme elas estiverem sendo referenciadas. Por isso, é possível executar a aplicação sem que se conheçam algumas classes.

O componente Javabean é distribuído através de um arquivo compactado com a extensão .JAR, denominado pacote, que contém, além de classes Java, arquivos de ícones e configurações. É possível embutir neste pacote classes para customização do componente. Os elementos de um pacote podem ser marcados como elementos de tempo de desenho, para que o mesmo não ocupe espaço na hora da distribuição.

Para (CRNKOVIC et al., 2007), a principal vantagem deste modelo de componentes é a sua simplicidade, porém se mostra insuficiente para grandes sistemas.

3.3.2 COM, DCOM, MTS e COM+

COM é a tecnologia desenvolvida pela Microsoft em 1995 para permitir a interoperabilidade de software escritas em linguagens distintas.

As interfaces dos componentes COM são especificadas através da linguagem Microsoft IDL, também conhecida como COM IDL. Há um identificador único para todo componente COM, denominado de GUID. Um componente COM pode possuir diversas interfaces.

O modelo de componentes COM define um protocolo que determina como objetos COM podem obter dinamicamente a interface de um outro componente. Isto é possível porque todo componente COM deve necessariamente herdar da interface IUnknown, que fornece métodos essenciais para qualquer componente, como por

exemplo, o método QueryInterface que é utilizado para extrair a interface de um componente.

Os componentes COM são distribuídos através de arquivos executáveis ou arquivos de biblioteca dinâmica (DLL).

Apesar do modelo de componentes COM estar disponível em diversas plataformas, o mesmo tem sido utilizado essencialmente nos sistemas operacionais da Microsoft.

O modelo de componentes DCOM é similar ao COM, porém permite distribuição remota e possui funções de segurança. O modelo MTS estende DCOM implementando recursos para persistência e transações.

3.3.3 CORBA

O modelo de componentes CORBA foi especificado pelo OMG, um consórcio de empresas de diversas áreas, tais como IBM, Philips, Canon. A idéia central de CORBA é permitir a criação e uso de objetos distribuídos, numa estrutura cliente/servidor, isto é, o servidor cria objetos e transfere a referência destes objetos para o cliente, para que este possa acessar os objetos remotamente.

A arquitetura CORBA é baseada num modelo de objetos abstratos que isola os clientes dos servidores. De acordo com (MAHMOUD, 1999), os objetos CORBA diferem da programação orientada a objetos convencional, pois:

- objetos CORBA podem executar em qualquer plataforma;
- objetos CORBA podem estar localizados em qualquer lugar numa rede de computadores;
- objetos CORBA podem ser escritos em qualquer linguagem que tenha um mapeamento IDL.

O modelo de componentes CORBA utiliza uma estrutura para gerenciamento de objetos distribuídos denominada Object Manager Architecture (OMA) e estão organizados conforme pode ser observado na Figura 7.

O ORB é responsável pela manipulação das requisições dos objetos, isto é, é ele que permite que objetos façam requisição e recebam requisição num ambiente heterogêneo de forma transparente. Por isso, o ORB se encarrega de localizar o

destinatário da requisição e de enviar os parâmetros necessários para o objeto no formato apropriado.

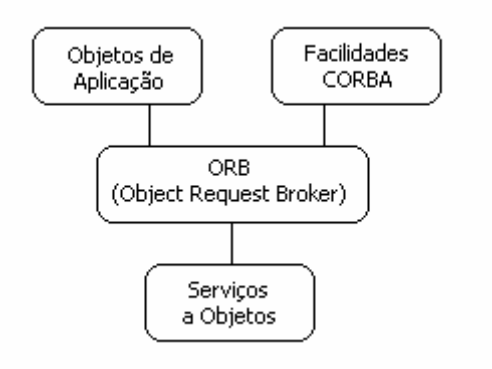

Figura 7 - Arquitetura CORBA

O componente "Objetos de Aplicação" representa todos os objetos acessíveis pela aplicação.

O componente "Serviços a Objetos" é responsável por disponibilizar uma série de serviços para controlar os objetos. Alguns dos serviços realizados por este componente são:

- controlar o ciclo de vida dos objetos: são serviços para criar, mover, remover e copiar objetos.
- segurança: disponibiliza serviços de autenticação, controle de acesso, criptografia e auditoria;
- persistência: oferece uma interface para que os objetos armazenem dados em bancos de dados ou sistema de arquivos.

As "Facilidades CORBA", são segundo (RICCIONI, 2000), coleções de serviços utilizados pelas mais diversas aplicações, como serviços para gerenciamento de informação, gerenciamento de sistema e inclusive serviços para áreas específicas, como contabilidade, medicina, etc.

O paradigma de plug-ins garante ao software maior capacidade de evolução e permite que a comunidade contribua para o seu desenvolvimento, características importantes para o software científico.

4 A TEORIA DA RESPOSTA AO ITEM

A Teoria da Resposta ao Item é utilizada para estimar variáveis latentes, isto é, estimar uma variável que não é diretamente observável. Uma variável latente é um conceito, tal como inteligência, auto-estima, cansaço, depressão, etc. Existe uma variedade de expressões utilizadas para referir-se a variável latente, conforme constatado por (PASQUALI, 2003), tal como traço latente, variável hipotética, construto, traço cognitivo, habilidade, aptidão, tendência e atitude. A variável latente é denotada na TRI através da letra grega teta (θ).

Apesar da variável latente não poder ser obtida por meios convencionais é possível estimá-la utilizando-se variáveis secundárias observáveis relacionadas à variável de interesse (ANDRADE et al., 2000).

Na maioria das vezes, para obter as variáveis observáveis, constrói-se um teste composto por um conjunto de itens (questões), onde cada item avalia um aspecto da variável latente de interesse. Enquanto que na Teoria Clássica de Testes (CTC), que é a forma tradicional de avaliação, a pontuação de um indivíduo (escore) é igual à quantidade de itens respondidos corretamente, na Teoria da Resposta ao Item, utiliza-se um modelo matemático para representar a relação das respostas com a variável latente e a partir daí, fazer inferências sobre os itens e a variável latente.

A Figura 8 ilustra a relação entre as respostas aos itens e uma variável latente. Assume-se que variações na variável latente causam variações nas respostas observadas, isto é, na pontuação de cada item.

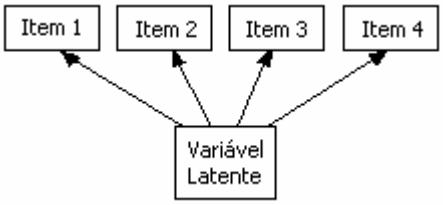

Figura 8 - Relação entre os itens e a variável latente

Alguns exemplos de aplicações da Teoria da Resposta ao Item são:

- personalização de ambiente de aprendizagem baseando-se na habilidade do indivíduo (CHEN et al., 2005);
- avaliar as atitudes dos empregados diante de uma corporação e suas políticas e procedimentos;

- identificar o nível de qualidade de vida de pacientes que estão em tratamento contra o câncer (GERSHON et al., 2003).

A TRI é utilizada especialmente na análise de testes educacionais. No Brasil, a TRI vem sendo empregada com sucesso na análise dos dados do Sistema Nacional de Ensino Básico (SAEB), conforme relatado por (ANDRADE et al., 2000).

A TRI, através do modelo matemático, relaciona as características dos itens - denominados de parâmetros dos itens - e as características dos indivíduos, com a probabilidade de uma resposta correta. (BAKER, 2001) destaca que, o interesse inicial da Teoria da Resposta ao Item é determinar se o indivíduo respondeu cada item individualmente de forma correta ou não, pois a ênfase da teoria é sobre os itens e não sobre o teste como um todo.

Alguns benefícios em se utilizar a Teoria da Resposta ao Item são:

- ao contrário da Teoria Clássica de Testes, que depende do conjunto de itens e das pessoas examinadas, os parâmetros dos itens e os parâmetros das pessoas são invariantes. Se um modelo da TRI fornece um bom ajuste para uma população como um todo, então um dado item terá os mesmos parâmetros em todos os sub-grupos desta população (HAMBLETON et al., 1991). Assim, as medidas obtidas a partir de uma análise da TRI não dependem da população na qual os dados foram extraídos;
- a TRI permite calcular o erro padrão de medidas baseadas na função de informação do teste e desta forma é possível selecionar itens que fornecem maior precisão de medidas para uma particular faixa de valores da variável latente;
- a TRI facilita o desenvolvimento de testes adaptativos de computador;
- a TRI possibilita a comparação entre populações diferentes, desde que tais populações tenham sido submetidas a itens comuns, ou até mesmo a comparação de indivíduos da mesma população que responderam testes totalmente diferentes (ANDRADE et al., 2000). A comparação se dá através de um processo denominado de equalização.

Para que seja possível realizar a análise através da TRI é necessário dispor dos parâmetros dos itens e o valor das variáveis latentes dos indivíduos, tarefa que é feita

através de um processo de estimação. O algoritmo utilizado para estimar os parâmetros é chamado de *método*.

Existe uma variedade de modelos matemáticos para diferentes tipos de testes. Cada modelo leva em consideração determinadas características dos itens.

4.1 Ferramentas da Teoria da Resposta ao Item

As principais ferramentas da TRI são denominadas de:

- Curva Característica de Item (CCI);
- Curva Característica do Teste;
- Função de Informação de Item (FII) e
- Função de Informação do Teste.

A Curva Característica do Item e a Curva Característica do Teste apresentam uma descrição detalhada do comportamento do item e do teste, enquanto que a Função de Informação do Item e a Função de Informação do Teste apresentam as precisões do item e do teste, respectivamente.

4.1.1 Curva Característica do Item

A relação entre a resposta dada por uma pessoa para um determinado item e a sua variável latente por ser representada graficamente através da Curva Característica do Item (CCI). A CCI demonstra a probabilidade de um indivíduo dotado de uma determinada variável latente responder corretamente a um item.

A partir da CCI é possível obter informações sobre os itens, porém tais informações são dependentes do modelo utilizado. Um exemplo de uma CCI pode ser visto na Figura 9:

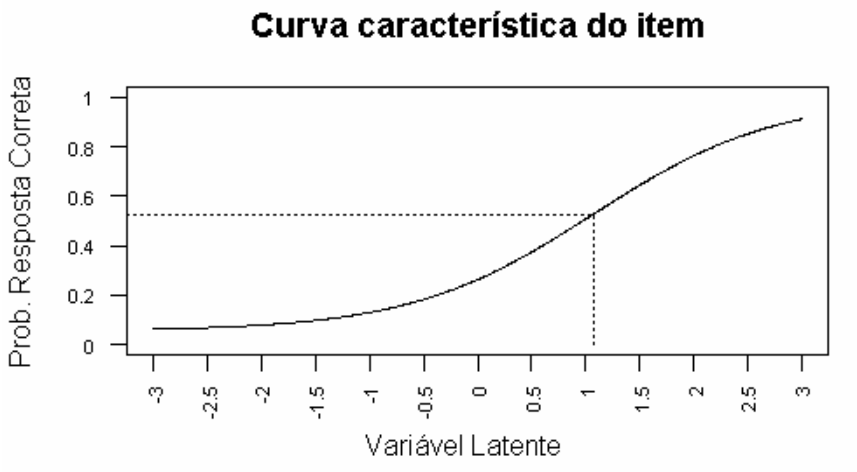

Figura 9 - Exemplo de Curva Característica de Item

O eixo vertical representa a probabilidade de acertar ao item e pode ser denotado por P(θ). Já o eixo horizontal indica o valor da variável latente. Para a medida "Variável Latente", de acordo com a Figura 9, quanto mais à direita, isto é, quanto maior o seu valor, maior será a probabilidade do indivíduo acertar o item. Desta forma, um indivíduo com variável latente igual a 3 tem probabilidade superior a 0,80 (ou 80%) em acertar tal item. Esta é uma suposição bem razoável, pois um indivíduo que é munido de um alto traço latente deve ter uma probabilidade maior de acertar ao item. O inverso também pode ser observado, isto é, um indivíduo com baixo traço latente (próximo de -3) tem uma pequena probabilidade em acertar ao item, ou seja, possui poucas chances de acertá-lo.

Por exemplo: supondo que se queira elaborar um teste para avaliar se um indivíduo possui ou não depressão (isto é, θ: depressão), seria construído um teste composto por um conjunto de itens de tal forma que as respostas aos itens pudessem chegar a esta conclusão. Um indivíduo com depressão teria uma alta probabilidade de responder que "na maior parte do tempo" se sente "desanimado e triste", enquanto que um indivíduo sem qualquer traço de depressão responderia que em nenhum momento se sente desanimado ou triste, ou pelo menos, somente em alguns poucos momentos se sente assim.

Teoricamente o intervalo da escala da variável latente é de $-\infty$ até $+\infty$, porém (BAKER, 2001) destaca que na maioria dos testes, a curva somente é descritiva para uma faixa restrita de níveis de habilidade.

4.1.2 Curva Característica do Teste

A Curva Característica do Teste é similar à CCI exceto que ela é obtida a partir de todo o conjunto de itens e por isso fornece informações sobre o teste como um todo. Na prática, é a soma de probabilidades computadas com a CCI de todos os itens e por isso é chamada de escore verdadeiro. Logo, dado um determinado valor para o traço latente, pode-se obter o valor do escore verdadeiro através da Curva Característica de Teste.

A expressão para calcular o escore verdadeiro é:

$$TS_j = \sum_{i=1}^{N} P_i(\theta_j)$$

onde: TS_j é o escore verdadeiro para os indivíduos com traço latente θ_j.

i representa o i-ésimo item

$P_i(\theta_j)$ é a probabilidade de que pessoas com traço latente θ_j respondam ao item *i* corretamente.

N é a quantidade total de itens.

O formato da Curva Característica do Teste depende do número de itens, do modelo matemático empregado e dos parâmetros dos itens.

A interpretação da Curva Característica do Teste é a mesma que a CCI, ou seja, quando maior o traço latente do indivíduo, maior a sua probabilidade de acertar os itens do teste e obter um bom escore verdadeiro.

De acordo com (BAKER, 2001), a principal função da Curva Característica do Teste é fornecer um meio de transformar o traço latente em escore verdadeiro. Esta é uma forma prática de interpretar o traço latente, afinal, este valor está relacionado com a quantidade de itens do teste. Para avaliação de desempenho educacional, por exemplo, o escore verdade pode ser utilizado como sendo a nota de um indivíduo num determinado teste. Além disso, uma outra função da Curva Característica do Teste é auxiliar o processo de equalização de testes.

4.1.3 Função de Informação do Item

O objetivo principal da teoria da resposta ao item é obter o valor da variável latente (θ). Como este valor não é observado diretamente, deve-se estimá-lo. O estimador da variável latente é denotado por $\hat{\theta}$. A função de informação do item permite

justamente calcular o grau de precisão do estimador da variável latente, isto é, o quão próximo do valor real o estimador está.

(ANDRADE et al., 2000) define a função de informação do item como sendo uma medida para analisar o quanto um item contém de informação para a medida da variável latente. Desta forma, quanto mais informação o item fornecer, mais preciso ou confiável é o estimador da variável latente e vice-versa.

A função de informação do item é calculada a partir da expressão:

$$I_i(\theta) = \frac{\left[\frac{d}{d\theta} P_i(\theta)\right]^2}{P_i(\theta) Q_i(\theta)}$$

onde: $\frac{d}{d\theta}$ é a derivada da função em relação a θ.

I é grau de informação do item i no nível θ do traço latente.

$Q_i(\theta)$ é igual a $1 - P_i(\theta)$.

A função de informação pode ser representada graficamente. Um exemplo pode ser visto na Figura 10:

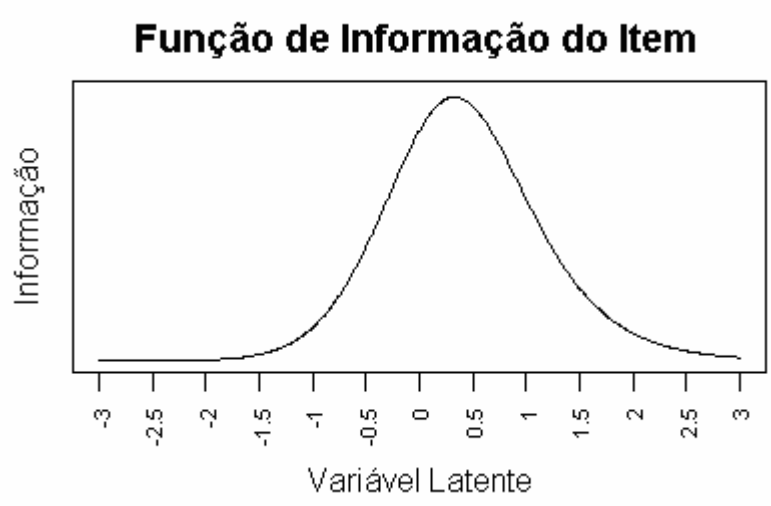

Figura 10 - Exemplo de Função de Informação do Item

O eixo horizontal representa a escala da variável latente e o eixo vertical representa a quantidade de informação que o item fornece. Para cada nível da variável latente é possível obter um determinado grau de informação, isto é, a quantidade de informação varia conforme o valor da variável latente. Nota-se na Figura 10 que o item é mais preciso para estimar a variável latente próxima a 0,5. Além disso, pode-se

observar que, quando a variável latente tem valores inferiores a -1 ou superiores a 1, o grau de informação fornecido por este item é muito baixo.

O desejável seria se o item pudesse fornecer o mesmo grau de informação em qualquer nível da variável latente, conforme apontado por (BAKER, 2001), porém na prática, isto não é possível de se conseguir. Logo, diferentes níveis da escala da variável latente possuem graus de precisão diferentes. Dependendo do propósito do teste, pode ser satisfatório obter bons graus de informação somente numa faixa do item.

A função de informação do item pode ser usada para identificar itens pobres ou pouco significativos para o teste. Para (REEVE & FAYERS, 2005), um item que tem um grau de informação baixo, pode significar que o item:

- mede algo diferente em relação aos outros itens na escala;
- não está bem elaborado e precisa ser reescrito;
- é muito complexo para os indivíduos;
- está fora do contexto do teste.

4.1.4 Função de Informação do Teste

De forma semelhante à Função de Informação do Item, a Função de Informação do Teste informa o grau de informação que o teste fornece para a medida da variável latente, e portanto, o quão preciso é o teste. Na prática, é a soma de informações dos itens para um determinado valor da variável latente, isto é:

$$I(\theta) = \sum_{i=1}^{N} I_i(\theta)$$

onde: $I(\theta)$ é a quantidade de informação do teste quando a variável latente for θ.

$I_i(\theta)$ é a quantidade de informação para o item *i* quando a variável latente for θ;

N é o número de itens do teste.

Em geral, quanto mais itens o teste possuir, maior deve ser a quantidade de informação do teste.

A função de informação do teste também pode ser plotada num gráfico. O exemplo da Figura 11 mostra que o teste é apropriado para estimar bem valores da variável latente no intervalo -2 a -0,5.

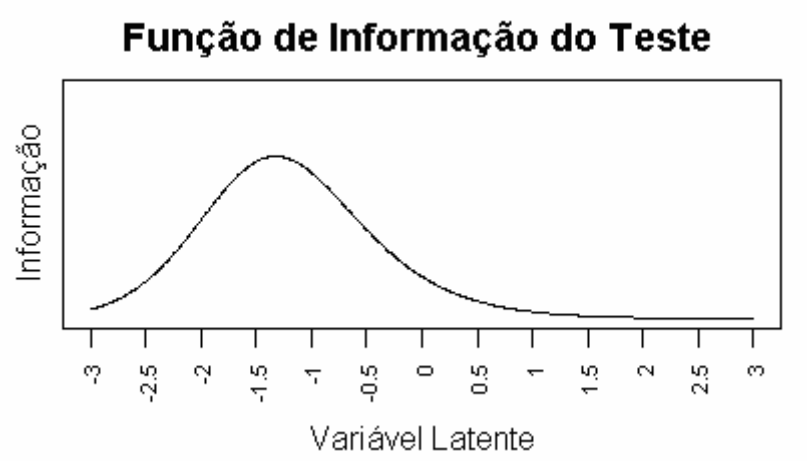

Figura 11 - Exemplo de Função de Informação do Teste

4.2 Modelos Matemáticos da TRI

Os modelos da TRI são usados para obter estimativas da variável latente, calibrar itens e examinar suas propriedades. Um modelo da TRI é uma função matemática que expressa a relação entre a probabilidade de um indivíduo com um determinado traço latente acertar um item, ou seja, o modelo é uma função entre os parâmetros do indivíduo e os parâmetros dos itens. Existem diversos modelos para a TRI que buscam representar esta relação. A quantidade de parâmetros e os seus significados dependem do modelo utilizado.

De acordo com (ANDRADE et al., 2000), os modelos da TRI são caracterizados por três fatores:

- de acordo com a natureza do item: Os modelos podem representar itens dicotômicos ou não dicotômicos. Itens dicotômicos podem ter somente uma resposta correta, enquanto itens não dicotômicos aceitam várias respostas corretas;
- número de populações envolvidas;
- quantidade de variáveis latentes que se deseja medir (uma variável ou várias).

Para aplicação adequada da TRI é esperado que o modelo represente bem os dados, isto é, que o modelo possa ser ajustado aos dados e que sejam atendidas as suposições impostas pelo modelo.

4.2.1 Modelos para itens dicotômicos

Os modelos mais comuns para análise de itens dicotômicos com a TRI são denominados de modelos logísticos de um, dois e três parâmetros. Estes modelos são aplicáveis quando:

- os itens estiverem medindo uma única variável latente. Em testes de conhecimento, (PASQUALI, 2003) relata que é sabido que o desempenho do ser humano é determinado por várias variáveis, porém existe uma variável que é dominante. O autor afirma que os desvios provocados pelo uso dos modelos logísticos são pequenos e podem ser ignorados, pois a solução dada pela TRI ainda é robusta. Este conceito chama-se de unidimensionalidade;
- as respostas a um item não podem interferir nas respostas de outros itens (independência local).

O modelo logístico de um parâmetro foi publicado na década de 1960 por George Rasch e por isso o modelo também é conhecido como modelo de Rasch (CHEN et al., 2005). Neste modelo os itens são caracterizados pela sua dificuldade e portanto, esta é a única propriedade que influência na determinação da variável latente do indivíduo. O parâmetro "dificuldade" geralmente é denotado pela letra *b*. A função deste modelo pode ser vista em:

$$P(\theta) = \frac{1}{1 + e^{-1(\theta - b)}}$$

Onde: b = dificuldade e

θ = valor da variável latente

O parâmetro *b* indica o ponto na escala da variável latente onde um indivíduo possui 50% de probabilidade de responder corretamente ao item. Quanto menor for o valor de *b*, mais fácil deve ser o item e quanto mais alto for seu valor, mais difícil deve ser o item.

A Figura 12 exemplifica duas curvas características de item do modelo logístico de um parâmetro. Observa-se que a curva tem sempre o mesmo grau de inclinação, o que muda, é a sua posição na escala. Conforme pode ser visto na CCI (a), um indivíduo com baixo valor da variável latente tem alta probabilidade de acertar ao item e por isso conclui-se que o item tem dificuldade baixa. Neste caso, para este item, o valor do

parâmetro *b* é -1,5. Já na CCI-b, para um indivíduo ter alta probabilidade de acertar ao item, também é necessário ter um alto valor para a variável latente e por isso conclui-se que este item tem dificuldade maior. Para este item, o valor do parâmetro *b* é 1,0.

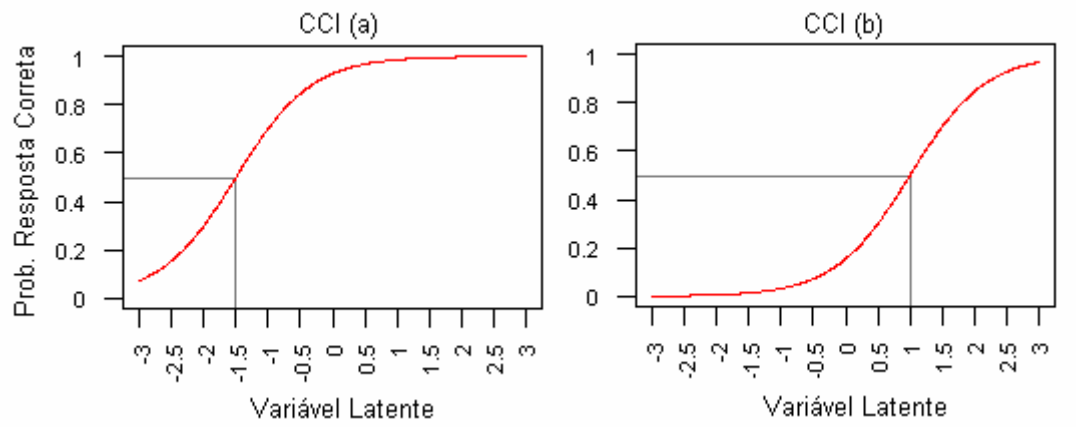

Figura 12 – CCIs do modelo logístico de um parâmetro

O modelo de dois parâmetros utiliza, além do parâmetro de dificuldade do item, um parâmetro adicional denominado de "discriminação". Este parâmetro é utilizado para indicar o quanto o item é capaz de discriminar indivíduos com variável latente inferior e superior à *b*. Geralmente utiliza-se a letra *a* para representar este parâmetro.

Sua função pode ser vista a seguir:

$$P(\theta) = \frac{1}{1 + e^{-a(\theta - b)}}$$

Onde: b = dificuldade e

a = é o parâmetro de discriminação

θ = valor da variável latente

A Figura 13 apresenta duas curvas características de item. Em ambos os casos, o valor de *b* é o mesmo (0,5). No primeiro caso (CCI-a), o valor de *a* é 0,8 e no segundo caso é 1,5. Quanto maior for o valor de *a* maior será o grau de inclinação da curva característica do item.

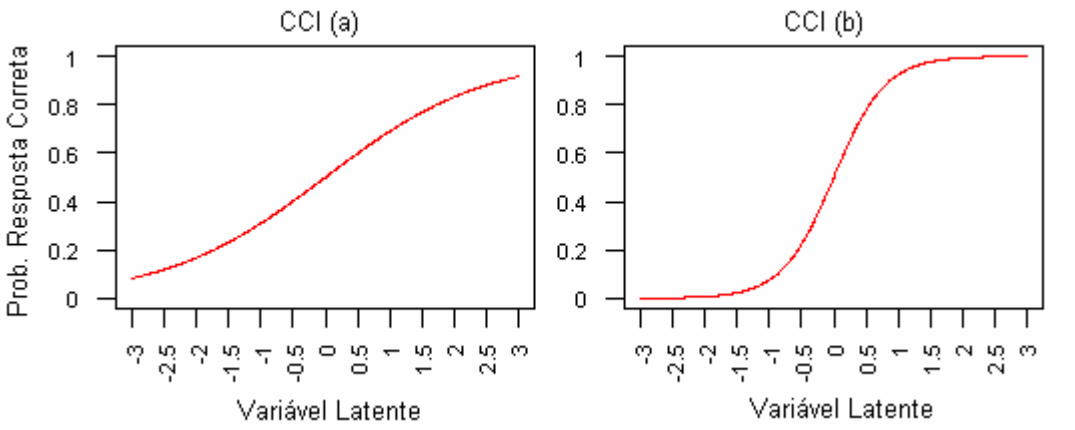

Figura 13 - CCIs do modelo logístico de dois parâmetros

Na maioria das vezes, os itens procuram representar a relação na qual, quanto mais o indivíduo for provido da variável latente, maior é a probabilidade do indivíduo acertar ao item. Entretanto, em alguns casos pode ocorrer o contrário, isto é, quanto maior for a variável latente do indivíduo, menor a probabilidade de acertar ao item. Neste caso, o valor do parâmetro *a* será negativo, indicando que o item foi mal elaborado e por isso deveria ser reformulado. A Figura 14 demonstra este caso, onde o valor da variável *a* é -1.

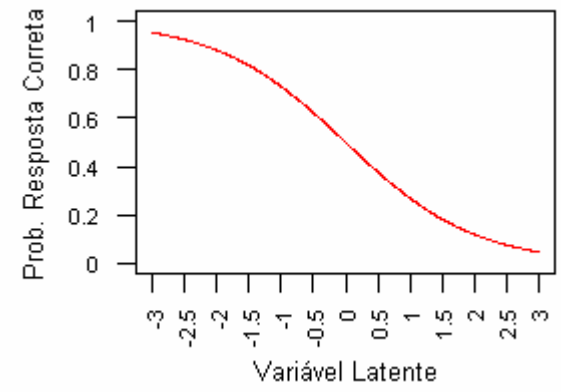

Figura 14 - CCI com discriminação negativa

Em 1968 Birbaun propôs a inclusão de um novo parâmetro para representar a contribuição de acerto ao acaso para a probabilidade de acerto (CHEN et al., 2005). Este modelo, conhecido como modelo logístico de 3 parâmetros ou modelo de Birbaun é aplicável quando os indivíduos têm probabilidade de acertar aos itens por casualidade. Neste caso, utiliza-se a letra *c* para denotar a probabilidade de acerto ao acaso. A função deste modelo pode ser vista em:

$$P(\theta) = c + (1-c)\frac{1}{1+e^{-a(\theta-b)}}$$

Onde: b = dificuldade e

a = é o parâmetro de discriminação

c = é o parâmetro de acerto ao acaso

θ = valor da variável latente

Conforme pode ser observado, o parâmetro *c* é constante para um item, isto é, independente do nível na escala da variável latente, seu valor sempre assume um único valor. Portanto, indivíduos com níveis distintos na escala da variável latente têm a mesma probabilidade de acertar um item ao acaso. Apesar do valor teórico do parâmetro c ser de 0 a 1, (BAKER, 2001) afirma que na prática este parâmetro atinge valores inferiores a 0,35.

Na Figura 15 estão demonstradas as curvas características de dois itens que foram analisados com o modelo de 3 parâmetros. Em ambos os itens, o valor do parâmetro *a* foi estipulado em 1,5 e o valor do parâmetro *b* em 1. Os itens diferem entre si em relação ao parâmetro *c*. Na CCI (a), o valor do parâmetro *c* é 0,1, enquanto que na CCI (b) o valor do parâmetro *c* é 0,3. Nota-se que o valor do parâmetro *c* define o limite inferior da CCI.

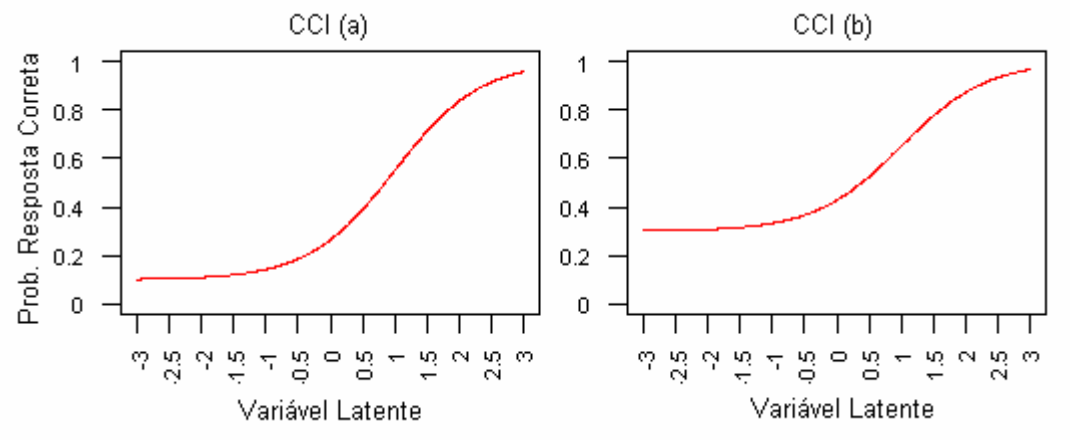

Figura 15 - CCIs do modelo logístico de dois parâmetros

(ANDRADE et al., 2000) destaca que todos os três parâmetros interferem na informação do item e seu valor será maior quando for possível observar que:

- *b* se aproxima de θ;
- *a* possuir um valor maior;

- quando *c* for mais próximo de 0.

4.2.2 Modelos para itens não dicotômicos

Os modelos que serão citados a seguir lidam com itens de resposta livre ou itens de múltipla escolha, na qual assume a existência de várias categorias de resposta, onde uma das categorias representa a escolha correta. Os modelos para itens não dicotômicos levam em consideração inclusive a escolha incorreta feita pelos indivíduos, aproveitando portanto esta informação para compor a análise. Os principais modelos são, de acordo com (ANDRADE et al., 2000):

- Modelo de Resposta Nominal: Trata-se de uma extensão do modelo logístico de dois parâmetros, porém o modelo foi adaptado para lidar com itens que tenham várias opções de escolha. Este modelo assume que não há uma seqüência (ordenação) entre as categorias de resposta. A probabilidade de resposta correta refere-se a cada categoria de resposta possível;
- Modelo de Resposta Gradual: Este modelo pode ser utilizado quando as categorias de resposta dos itens estão dispostos em ordem gradual, como por exemplo, com opções de resposta: discordo totalmente, discordo, concordo e concordo totalmente. O modelo utiliza um parâmetro de dificuldade para cada categoria de resposta;
- Modelo de Escala Gradual: Semelhante ao modelo anterior, porém este modelo assume que todos os itens compartilham as mesmas categorias e a mesma escala de classificação;
- Modelo de Crédito Parcial: Trata-se de uma extensão do modelo de Rash porém adaptado para lidar com itens não dicotômicos. Este modelo assume que todos os itens têm a mesma capacidade de discriminação;
- Modelo de Crédito Parcial Generalizado: Baseia-se no modelo de créditos parcial, porém é mais flexível quanto à hipótese de poder de discriminação uniforme para todos os itens.

4.3 Estimação

Na maior parte das vezes, tanto os parâmetros dos itens como a variável latente são desconhecidos, sendo necessário a execução de um processo para obter seus valores. Em alguns casos entretanto, os parâmetros dos itens podem ser conhecidos, sendo necessário calcular apenas a variável latente, ou então, a variável latente dos indivíduos é conhecida e é preciso apenas calcular os parâmetros dos itens. Em qualquer caso, o processo utilizado para obter os parâmetros dos itens ou a variável latente dos indivíduos é denominado de estimação. A estimação de parâmetros dos itens também é conhecida como Calibração.

Supondo que sejam conhecidos os parâmetros dos indivíduos e as respostas dadas a um determinado item, é possível observar os dados conforme apresentado na Figura 16. Nesta figura cada ponto representa a proporção de respostas corretas dadas por indivíduos com uma determinada faixa de variável latente.

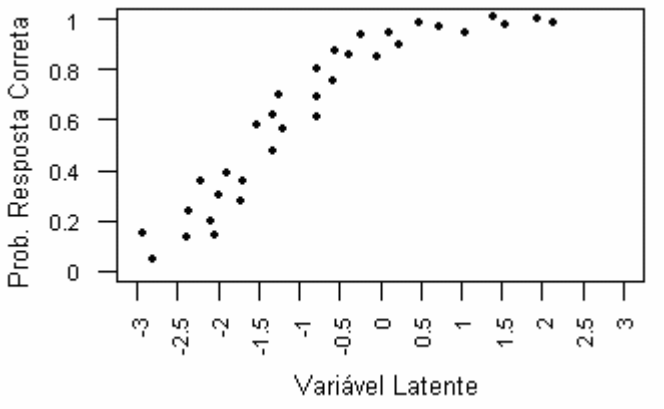

Figura 16 - Proporção de respostas corretas por faixa de variável latente

Para estimar os parâmetros deste item, deve-se primeiro determinar que modelo se ajusta melhor aos dados, isto é, que modelo utilizar para representar a proporção observada de respostas corretas em relação à variável latente. Em seguida utiliza-se um método matemático capaz de estimar os parâmetros dos itens. Um dos métodos mais conhecidos na TRI é o método da máxima verossimilhança (MV), através do algoritmo Newton-Raphson, que inicia o processo estabelecendo valores arbitrários para os parâmetros do item. O próximo passo é a utilização da função do modelo escolhido para calcular o valor de $P(\theta)$ para todos os níveis da variável latente e o valor da função de verossimilhança. As estimativas dos parâmetros dos itens são os valores que

maximizam esta função. A obtenção deste valores, em geral, requerem algoritmos iterativos. No final do processo tem-se estimados os parâmetros do item e é possível construir a curva característica do item da curva ajustada, conforme mostrado na Figura 17.

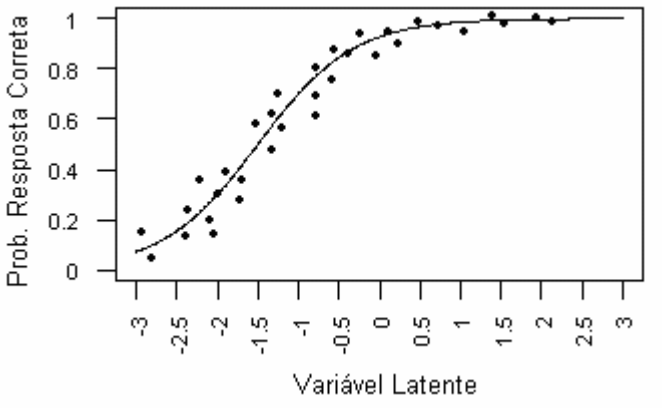

Figura 17 – Exemplo de modelo ajustado

A situação mais comum na TRI é desconhecer os parâmetros dos itens e o valor da variável latente dos indivíduos, exigindo portanto, a estimação de ambos os parâmetros. Existem duas estratégias para estimar os parâmetros neste caso: estimação conjunta ou estimação de itens seguida da estimação dos valores da variável latente.

Na estimação conjunta, há o processo proposto por Birnbaun na qual assume inicialmente valores padronizados para a variável latente e por isso a estimação de parâmetros de itens assume que o valores da variável latente sejam conhecidos. Em seguida, estimam-se os valores da variável latente a partir da estimativas dos parâmetros dos itens obtidas anteriormente. Este processo é executado iterativamente até que seja atingido um critério de parada. Este método sofre de um problema de falta de consistência quando estão envolvidos muitos indivíduos ou muitos itens, conforme observado por (ANDRADE et al., 2000). Neste caso, um método alternativo é o método de Máxima Verossimilhança Marginal que considera a existência de uma distribuição associada à habilidade dos indivíduos.

Existem várias outras propostas para estimação dos parâmetros dos itens e dos indivíduos.

O processo de estimação envolve procedimentos matemáticos avançados e por isso requer o uso intensivo do processador do computador. Conforme (BAKER, 2001), a Teoria da Resposta ao Item somente passou a ser utilizada devido a grande

disponibilidade de computadores e atualmente o processo de estimação de parâmetros já não é mais um problema para os computadores atuais. Uma descrição detalhada de métodos de estimação pode ser encontrada em (ANDRADE et al., 2000).

4.4 Equalização

A equalização refere-se a um processo para permitir que testes possam ser comparados. Para que a comparação possa ser realizada é necessário que os parâmetros dos itens dos testes estejam na mesma métrica. Da mesma forma, para comparar a variável latente de indivíduos que responderam testes diferentes é necessário que os valores estejam na mesma escala. O processo de equalização garante que os dados a serem comparados estejam na mesma escala.

Existe uma variabilidade de comparações que podem ser dirigidas, dependendo da quantidade de grupos de indivíduos e testes envolvidos, conforme relata (ANDRADE et al., 2000):

- um único grupo fazendo dois testes totalmente diferentes;
- um único grupo fazendo dois testes parcialmente diferentes;
- dois ou mais grupos fazendo um único teste;
- dois ou mais grupos fazendo dois testes parcialmente diferentes.

Para que seja possível equalizar dois testes, é necessário que pelo menos eles tenham algo em comum. Não é possível fazer uma comparação entre grupos distintos fazendo dois testes diferentes.

Quando os itens dos testes são novos, isto é, são itens não calibrados, a equalização se dá através da calibração simultânea dos parâmetros dos itens e/ou habilidades dos diversos testes.

Há uma abordagem alternativa, denominada Equalização a Posteriori, que é aplicada quando se lida com grupos diferentes. Neste caso, os parâmetros dos itens de cada grupo são estimados separadamente. Como os testes devem ter alguns itens em comum, estes itens terão parâmetros diferentes em cada teste. A equalização parte disto para estabelecer uma relação entre os testes, utilizando-se do princípio da invariância dos itens.

Conforme pode ser visto, a TRI permite obter informações mais abrangentes que a Teoria Clássica de Testes, razão pela qual (HARVEY & HAMMER, 1999) prevêem que no futuro métodos baseados da TRI substituirão os métodos baseados na Teoria Clássica de Testes.

5 DESENVOLVIMENTO DO AMBIENTE

Conforme apresentado nos capítulos anteriores, a TRI exige a execução de algoritmos sofisticados e por isso a importância do computador para realizar tal processamento. Além disto, sabe-se que os modelos e métodos da TRI estão em constante evolução, por isso representam o foco principal do estudo da TRI. O ambiente proposto neste trabalho foi elaborado com intuito de atender a utilização da Teoria da Resposta ao Item levando-se em consideração os seguintes aspectos:

- permitir ao avaliador processar a análise de testes através da TRI. Consiste em gerar relatórios e gráficos apresentando a análise realizada dos testes que foram aplicados;
- permitir o desenvolvimento de novos métodos da TRI. Visto que os testes podem ser modelados de diversas formas, de acordo com a natureza dos seus itens, é esperado que o ambiente seja capaz de incorporar novos métodos da TRI. Esta característica favorece tanto os avaliadores que aplicam a TRI para análise de testes quanto aos pesquisadores da TRI. O ambiente proposto poderá ser utilizado pelos pesquisadores da TRI para criar novos modelos seguindo uma especificação que busca padronizar e facilitar a criação de métodos da TRI. Já para os avaliadores, esta característica facilitará o acesso às pesquisas mais recentes com a TRI.

Para que o ambiente suportasse a incorporação de novas implementações de métodos da TRI sem que fosse necessário recompilar o projeto, o software foi construído usando o conceito de plug-ins. Cada método da TRI é implementado e distribuído como um plug-in. A linguagem R foi adotada para o desenvolvimento dos métodos da TRI, já que R é considerada uma linguagem popular entre os pesquisadores da TRI. Desta forma, todos os plug-ins são implementados nesta linguagem.

Além dos plug-ins de métodos da TRI, existe uma outra categoria de plug-ins (denominada “Geral”) suportada pelo ambiente que é capaz de executar qualquer outro programa R. Esta categoria de plug-ins foi construída para oportunizar o desenvolvimento de programas para orientar o usuário Avaliador.

O diagrama da Figura 18 representa a organização dos componentes utilizados para construção do ambiente.

O componente “Software TRI” disponibiliza a interface com usuário, sendo portanto, o meio pelo qual o usuário interage com o ambiente. A solicitação de análise de testes e a exibição dos seus resultados ocorrem através desta parte do software. Para iniciar uma análise com a TRI o usuário deve fornecedor os dados, isto é, a relação de indivíduos, os itens do teste e as respostas dadas pelos indivíduos para os itens. Além disso, é necessário que seja informado qual método deve ser utilizado para processar a análise. Ao selecionar o método a ser utilizado, o software encarrega-se de instanciar o plug-in apropriado

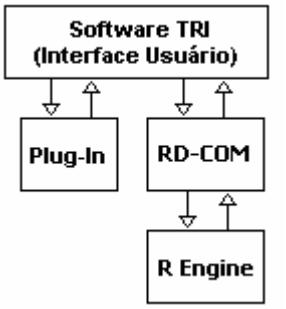

Figura 18 - Diagrama dos componentes usados do Ambiente

Toda análise da TRI está implementada em R, através de plug-ins. Como a linguagem R é interpretada, os plug-ins não são constituídos de programas em linguagem de máquina. Cada plug-in é distribuído com todos os códigos-fonte em R, necessários para sua execução. Apenas os pacotes R externos não são distribuídos com o plug-in, pois precisam ser instalados previamente. Um plug-in de método da TRI possui também toda a declaração da sua interface de programação, para que o ambiente seja capaz de saber como interagir com tal plug-in. Desta forma, o ambiente é dotado de um mecanismo para execução de programas R, isto é, o “Software TRI” encarrega-se de instanciar o plug-in a fim de obter os programas a serem executados, conduzindo-os em seguida para um Engine R, que é capaz de efetivamente processar os programas R.

A comunicação entre o “Software TRI” e o Engine R é feito através do componente RD-COM. O componente RD-COM contém um servidor DCOM que é utilizado para conectar “aplicações clientes” com o Engine R. O componente RD-COM disponibiliza uma interface de acesso ao R através de objetos COM e controles ActiveX (tecnologia criada pela Microsoft para facilitar a integração entre diversas aplicações).

O “Software TRI” cria uma seção no Engine R assim que algum programa R precisar ser executado. Esta seção permanece ativa enquanto o “Software TRI” permanecer em execução, por isso, novas requisições ao Engine R não precisam reestabelecer uma nova seção.

5.1 Apresentação do ambiente

A Figura 19 exibe a tela inicial do software. Os recursos ou funções são acionados a partir das opções dispostas no menu lateral esquerdo que está organizado em grupos. O grupo denominado “TRI” é o único grupo de funções que está embutido no software. Outros grupos de funções podem ser adicionados a partir de plug-ins da categoria “Geral”. O método utilizado pelo ambiente para localização de plug-ins da categoria “Geral” é pesquisar o sistema de arquivos no diretório “plug-ins” que deve estar no mesmo diretório do arquivo executável. No exemplo da Figura 19, o grupo “Básico TRI” foi adicionado através de plug-ins da categoria “Geral”.

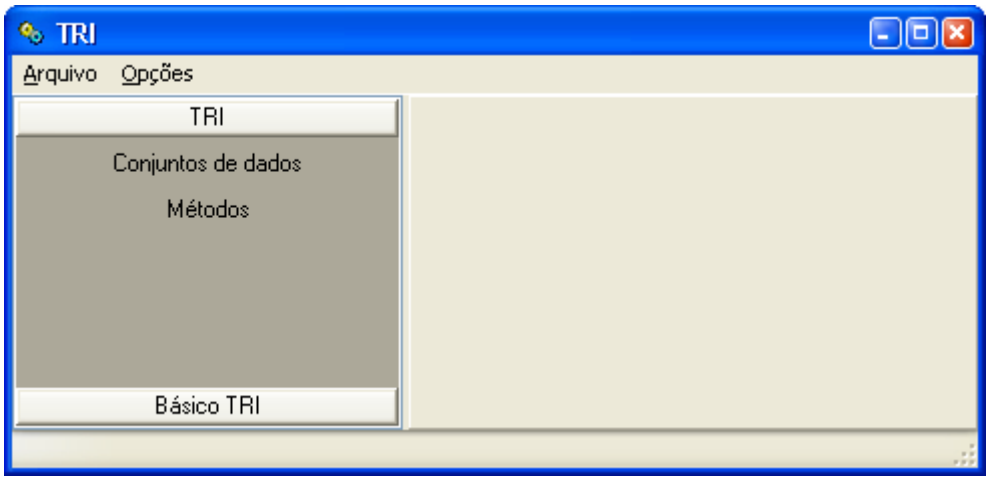

Figura 19 - Tela inicial do ambiente

A função “Métodos” do grupo “TRI” permite a instalação de novos métodos da teoria da resposta ao item, além da visualização dos métodos já instalados (Figura 20). A instalação também pode ser feita armazenando o plug-in no sub-diretório “methods” residente no diretório do arquivo executável.

Os plug-ins que são copiados diretamente pelo sistema de arquivos, quer sejam plug-ins da categoria “Geral” ou da categoria “Métodos da TRI”, somente estarão disponíveis para execução depois do ambiente ser reinicializado.

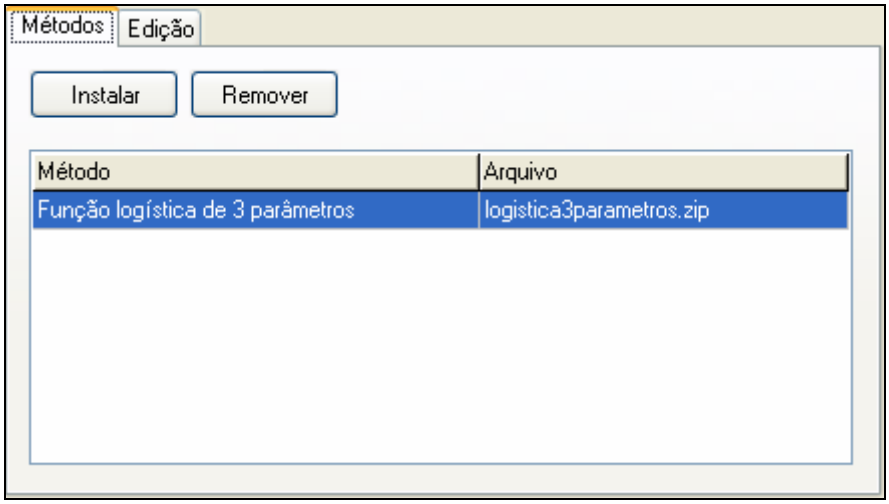

Figura 20 - Tela para instalação de métodos

A função "Conjunto de dados" do grupo "TRI" permite realizar as operações para análise de testes utilizando a teoria da resposta ao item (Figura 21).

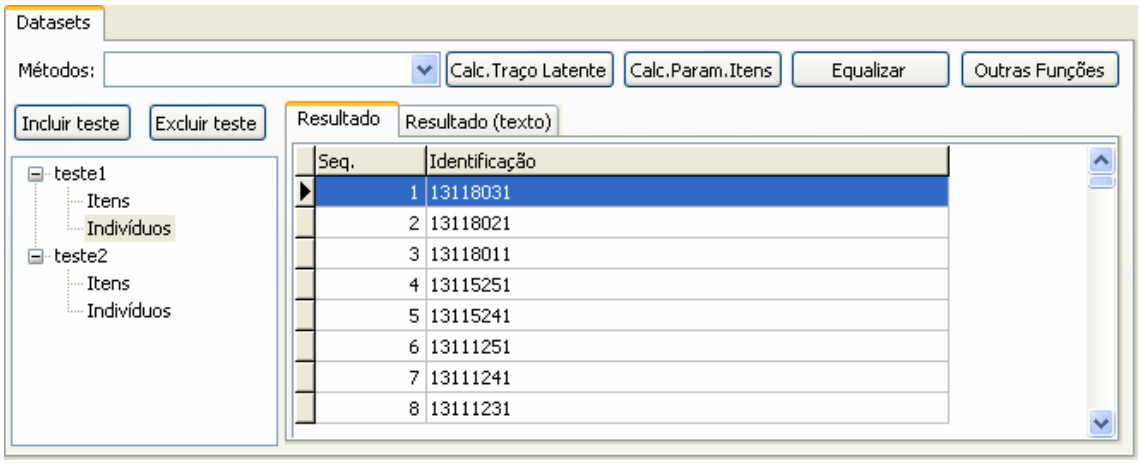

Figura 21 - Tela para análise de testes

Os botões "Incluir teste" e "Excluir teste" são utilizados para adicionar um teste e remover um teste, respectivamente. Normalmente os dados referentes aos testes são disponibilizados em arquivos textos pré-formatados. Ao solicitar a inclusão de um novo conjunto de dados através do botão "Incluir teste", o software solicita ao usuário que seja informada a configuração da formatação do arquivo.

Em seguida, para prosseguir com a análise do teste, deve-se selecionar o método a ser utilizado. A caixa de seleção "Método" exibe os nomes de todos os métodos instalados. Uma vez adicionado um ou vários testes e selecionado o método, pode-se iniciar o processo de análise que pode ser a partir do cálculo dos parâmetros dos itens ou cálculo do traço latente. Para executar qualquer um destes processos é necessário fornecer alguns parâmetros para a realização dos cálculos de estimação, portanto, antes

de prosseguir é necessário informar os valores dos parâmetros. O software já exibe na tela valores padrões para os parâmetros sendo necessário apenas informar aqueles parâmetros que são diferentes do usual. A Figura 22 mostra uma tela de exemplo que é exibida ao solicitar o cálculo de Parâmetros de Itens de um método.

Ao confirmar, o software realiza os cálculos de estimação. Todos os processos são realizados através da execução de programas escritos em linguagem R, portanto, neste instante, faz-se conexão com o Engine R.

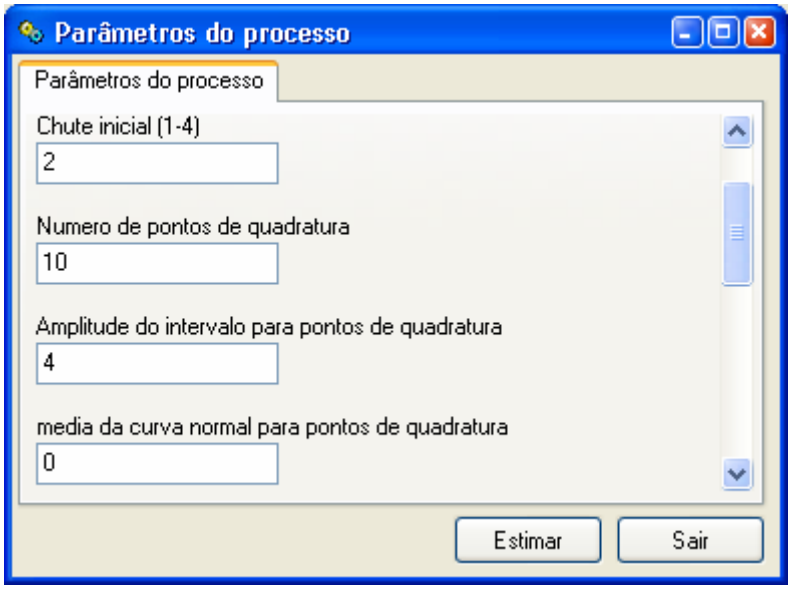

Figura 22 - Tela para digitação de parâmetros para cálculo

Depois de realizado o processo de cálculo de parâmetros dos itens ou o cálculo da variável latente, pode-se visualizar os resultados no Ambiente. A Figura 23 mostra a exibição dos parâmetros dos itens identificados pelo cálculo.

Resultado | Resultado (texto)

Seq.	Identificação	a	b	c
1	29	1,1822	1,0698	0,0539
2	23	1,6397	-1,0944	0,0378
3	31	1,5270	-1,2348	0,3546
4	32	1,5052	-0,8731	0,1827
5	33	1,2259	-0,5388	0,0350
6	34	1,5030	-1,6061	0,4809
7	35	1,5097	-1,8333	0,4619
8	36	1,4469	-2,0918	0,3959
9	24	0,9167	0,1029	0,0669
10	11	1,7950	-0,7994	0,0235
11	13	1,0978	-0,4316	0,0648
12	14	1,0000	0,8056	0,2500

Figura 23 - Exemplo de saída de resultados

Além disso, o usuário tem acesso às ferramentas da TRI, tal como a curva característica e a função de informação de cada item, o escore verdadeiro e a função de informação do teste. Todas estas ferramentas de análise da Teoria da Resposta ao Item estão disponíveis a partir do botão "Funções", que são parte integrante dos plug-ins. A Figura 24 exibe a tela de funções da teoria da resposta ao item para um determinado método selecionado.

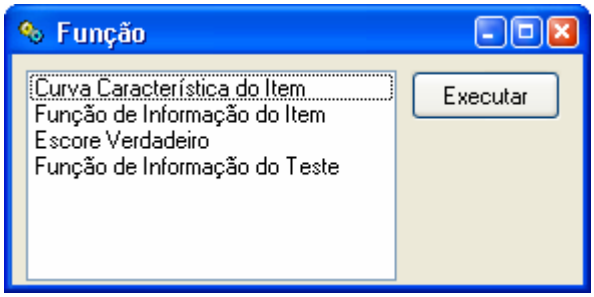

Figura 24 - Tela para seleção de função da TRI

Outras informações adicionais também podem ser obtidas clicando-se nas opções "Itens" ou "Indivíduos" do teste realizado.

O botão "Equalizar" permite ao usuário realizar uma análise comparativa entre dois ou mais testes. Para tanto é necessário informar quais testes se deseja comparar e que comparação se pretende realizar (comparação de parâmetros de itens ou comparação de habilidades dos respondentes), conforme exibido na Figura 25:

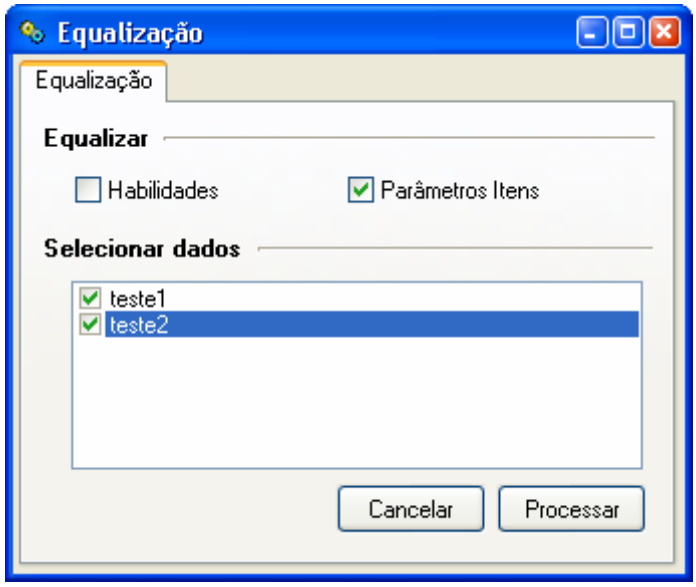

Figura 25 - Tela para seleção de dados para equalização

5.2 Especificação do software

A Figura 26 apresenta os principais elementos da especificação em UML para o desenvolvimento do ambiente.

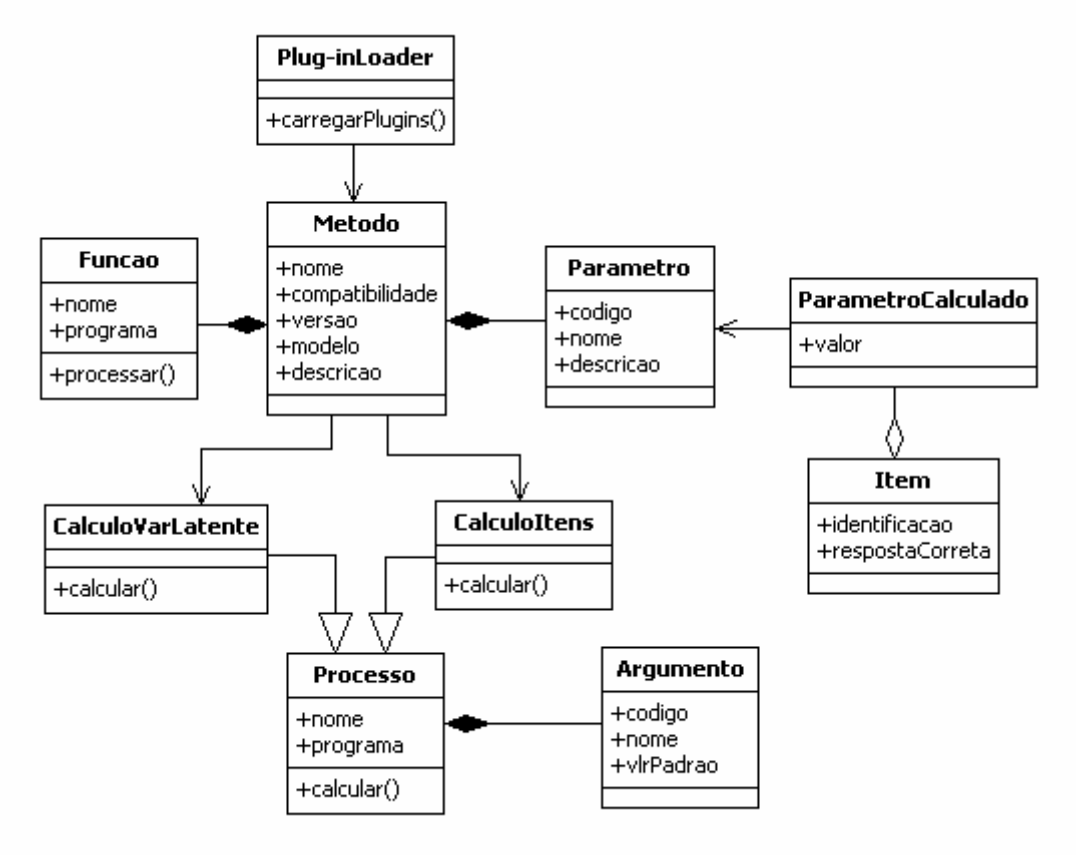

Figura 26 - Diagrama de Classes da categoria "Métodos"

A classe "Plug-inLoader" tem a mesma finalidade da classe "Plug-in Loader" sugerida por (MAYER et al., 2002), isto é, esta classe é responsável por localizar no sistema de arquivos todos os plug-ins existentes, quer sejam plug-ins de Métodos ou plug-ins Gerais.

A classe "Metodo" representa um método da teoria da resposta ao item, sendo responsável por encapsular as implementações necessárias para o cálculo da variável latente dos indivíduos e pelo cálculo dos parâmetros dos itens. A propriedade "compatibilidade" foi definida para informar ao ambiente a versão da especificação na qual o método é compatível, a fim de evitar resultados inesperados quando executado em ambiente de versão imprópria. Além disso, há a propriedade "versao" que é utilizada para identificar a versão do método implementado.

Um Método é composto por uma instância de CalculoVarLatente e CalculoItens, que são as classes que se encarregam de calcular a variável latente dos indivíduos e os parâmetros dos itens, respectivamente. Ambas as classes são heranças da classe

Processo. Uma chamada ao método Processar desta classe executa a seqüência de operações descrita na Figura 27.

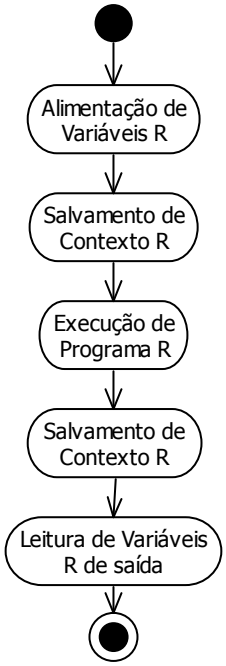

Figura 27 – Execução de programa R

Num primeiro momento ocorre a alimentação de variáveis R de acordo com a interface exigida pelo cálculo. Esta atividade consiste em ler a interface do programa a ser executado, solicitar ao usuário os dados exigidos pelo programa e enviá-los para o Engine R. Após esta etapa, ocorre o salvamento do contexto R. Em seguida, ocorre a execução de programa R para efetuar os cálculos apropriados. Após a execução do algoritmo, ocorre novamente o salvamento do contexto R. Por último, faz-se a leitura de variáveis R resultantes da execução do programa.

A classe “Argumento” foi especificada para manter os dados necessários para que seja possível solicitar ao avaliador os argumentos ou parâmetros para cálculo e análise do teste. Os argumentos de cálculo são informações exigidas pelo algoritmo do método com intuito de configurar como a análise deve ser realizada. Estas informações devem constar na especificação do plug-in. Cada instância da classe Argumento permite a exibição na tela de um campo para digitação com respectivo nome e variável R associada. Opcionalmente pode ser informado um valor padrão para o argumento, através da propriedade “vlrPadrao”. Quando for informado algum dado no campo, este é

enviado para o programa codificado na linguagem R. O ambiente encarrega-se de criar uma variável no ambiente R utilizando como seu nome o valor do atributo "codigo" e definindo seu valor de acordo com o valor digitado pelo avaliador. Este é o mecanismo utilizado para estabelecer a transferência de informações entre a aplicação e os programas codificados na linguagem R. Tanto o cálculo de parâmetros dos itens como o cálculo da variável latente contam com este recurso. Imediatamente depois de criar as variáveis no Engine R, ocorre o salvamento do contexto R no arquivo before.Rdata. Em seguida, o método Processar da classe Processo, executa o programa informado na propriedade "programa". Depois de concluída a execução do programa, salva-se o contexto do Engine R novamente, agora no arquivo after.Rdata. Os procedimentos para salvar o contexto foram construídos para auxiliar o desenvolvimento e depuração de implementações de novos métodos da TRI. Ambos os arquivos são gravados no mesmo diretório do aplicativo principal. Os nomes dos arquivos (before.Rdata e after.Rdata) estão fixados e não podem ser alterados pelo usuário. No Console R, para carregar um contexto, basta executar a instrução *load* informando como parâmetro, o arquivo que armazena o contexto.

Além de enviar para o Engine R dados digitados pelo usuário para configuração do cálculo, a classe Processo também disponibiliza uma série de estruturas necessárias para o processamento dos cálculos. Estas estruturas serão objetos R acessíveis pelo programador para codificação do método em R. O programa ReadTest.r, que deve constar no mesmo diretório do programa executável do software TRI, encarrega-se de interpretar as informações do arquivo de dados e definir estas estruturas. A relação de estruturas e seus respectivos tipos de dados e descrição constam na Tabela 1.

Tabela 1 - Estruturas disponibilizadas para programas R

Objeto	Tipo de dado	Descrição do conteúdo
dataset.original	Matriz	Todos os dados da análise (isto é, a relação dos alunos, as respostas das questões e o gabarito)
dataset	Matriz	Identificação dos alunos e suas respectivas respostas
dataset.info	Matriz	Informações a respeito do gabarito e dados sobre as questões.
items.rightanswers	Vetor	Gabarito das questões
items.numoptions	Vetor	Quantidade de opções (alternativas de escolha) para cada questão
items.deal	Vetor	Deverá manter "S" se a questão deverá ser processada ou "N" se deve ser ignorada.
items.name	Vetor	Identificador das questões
dataset.answers	Matriz	Respostas dadas pelos alunos.

A Figura 28 apresenta um exemplo de um arquivo de dados, onde estão destacadas as estruturas criadas pelo ambiente.

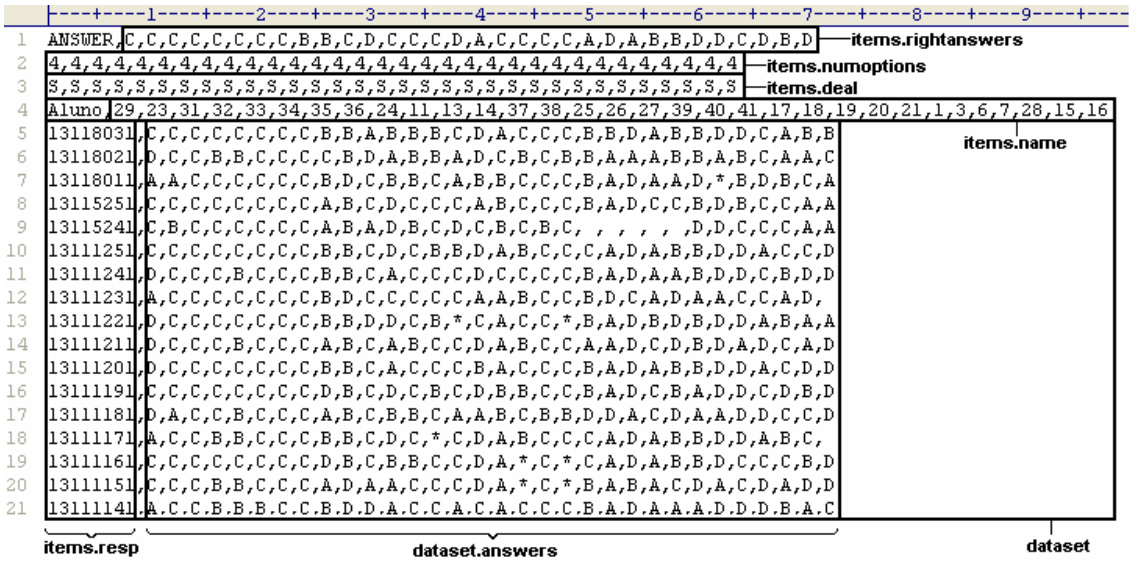

Figura 28 - Arquivo de dados exemplo

Durante a execução do programa de cálculo da variável latente é esperado que uma estrutura denominada “abilities” seja alimentada com as variáveis latentes calculadas para os indivíduos. Após efetuar este processamento, o software constrói um relatório de análise baseando-se nas informações contidas nesta estrutura. A estrutura abilities é um vetor onde o número de linhas é equivalente ao número de indivíduos.

Algo semelhante também é realizado durante o cálculo das estimativas dos parâmetros dos itens: os valores calculados pelo método devem ser armazenados numa estrutura denominada “items.estimatives”. Esta estrutura é uma matriz onde o número de linhas é equivalente ao número de indivíduos e o número de colunas é igual à quantidade de parâmetros utilizados pelo método. O ambiente cria um relatório apresentando as análises realizadas.

A classe “Funcao” permite a representação de alguma ferramenta da teoria da resposta ao item, tal como a curva característica do item e a função de informação do teste. Estas ferramentas da TRI são executadas a partir de programa escrito em linguagem em R e por isso o atributo “programa” é utilizado para armazenar o nome do programa R principal. Algumas funções são realizadas para um determinado item enquanto que outras são realizadas para um determinado indivíduo. Há também o caso em que alguma função será executada porém o processamento se dará sobre todo o conjunto de dados.

A classe “Item”, usada para representar um item, possui uma agregação com a classe “ParametroCalculado”, que é utilizada para armazenar o valor calculado para um parâmetro de um determinado item para um certo método.

5.3 Construção de novos métodos

Para construir um novo método e distribuí-lo para que outros usuários possam fazer uso de tal método é necessário construir um plug-in.

Um “plug-in de método” é um pacote que contém a implementação de um método da teoria da resposta ao item. O plug-in é um arquivo compactado (no formato .zip) composto por arquivos de programas em linguagem R e a publicação da interface do plug-in, que é armazenada em arquivos XML.

Todo pacote de método deve possuir o arquivo index.xml, que é utilizado para descrever os componentes necessários para execução do método. Index.xml é um arquivo estruturado, que contém as seguintes informações:

- o nome do método;
- O nome do programa principal responsável por realizar o cálculo dos parâmetros dos itens;
- O nome do arquivo que contém a descrição da interface de cálculo dos parâmetros dos itens;
- O nome do arquivo que contém a interface de cálculo da variável latente;
- Os nomes dos parâmetros dos itens;
- O nome do programa responsável por realizar o cálculo da variável latente;
- O nome do programa responsável por realizar o cálculo de equalização;
- Os nomes das funções e os respectivos nomes dos arquivos capazes de executá-los;

A descrição no formato XML Schema do arquivo index.xml consta no Anexo 1.

Um exemplo de um arquivo index.xml pode ser visto no Quadro 3.

Quadro 3 - Exemplo de descrição de interface de plug-in

```
<method>
  <name>Funcao logistica de 2 parametros</name>
  <parameters>
    <program>parametrositens.r</program>
    <input>inputpar.xml</input>
    <output>
      <name>Curva Caracteristica do Item</name>
      <program>cci.r</program>
```

```
    </output>
    <name>a</name>
    <name>b</name>
  </parameters>
  <abilities>
    <program>habilidades.r</program>
    <input>inputhab.xml</input>
    <output>
      <name>Escore Verdadeiro</name>
      <program>escoreverdadeiro.r</program>
    </output>
  </abilities>
  <equating>
    <program>equalizacao.r</program>
  </equating>
  <output>
    <name>Funcao de Informacao do Teste</name>
    <program>funcaoinformacaoteste.r</program>
  </output>
  <compability>1.0</compability>
</method>
```

O arquivo index.xml apresentado no Quadro 3 fornece as seguintes informações:

- O nome do método é: “Função logística de 2 parâmetros”
- O programa responsável por realizar a estimativa dos parâmetros dos itens é “parametrositens.r”;
- A descrição da interface para inserção de argumentos para cálculo dos parâmetros dos itens está armazenada no arquivo inputpar.xml;
- O programa responsável por exibir a curva característica do item é “cci.r”;
- O método utiliza dois parâmetros, denominados “a” e “b”;
- Há uma função que é executada para cada indivíduo cujo nome é “Escore Verdadeiro” e o programa que o realiza é “escoreverdadeiro.r”;
- A descrição da interface para inserção de argumentos para cálculo das habilidades está armazenada em inputhab.xml;
- O programa responsável por realizar a estimativa das habilidades é “habilidades.r”;
- Para comparação entre dois ou mais testes utiliza-se o programa “equalizacao.r”;
- Existe uma função que é executada para o teste chamada “Função de informação do Teste” e o programa que implementada esta função está armazenado em “funcaoinformacao.r”;
- Este método é compatível com a versão 1.0 da especificação.

Todos os arquivos mencionados no arquivo index.xml devem estar armazenados no pacote.

Os programas para cálculo de parâmetros de itens bem como os programas para cálculo de variável latente exigem que o usuário forneça algumas informações antes de prosseguir com o cálculo. Estas informações são chamadas de argumentos do processo e devem estar armazenadas no pacote. A descrição do arquivo XML no formato XML Schema pode ser visto no Anexo 2.

Um arquivo de exemplo pode ser visto no Quadro 4. Neste exemplo, estão sendo definidos dois argumentos para execução de um determinado programa:

- Número máximo de interações do algoritmo Newton Raphson
- Critério de parada

Quadro 4 - Exemplo de declaração de interface de programa

```
<variables>
  <variable>
    <name>tmax</name>
    <label>Nr. máximo de iteraçõe do algoritmo Newton Raphson</label>
    <default>50</default>
  </variable>
  <variable>
    <name>crit</name>
    <label>Critério de parada</label>
    <default>0.01</default>
  </variable>
</variables>
```

A tela exemplificando a solicitação destes argumentos pode ser vista na Figura 29. A propriedade default informa neste caso o valor padrão de cada argumento. Desta forma, o valor “0.01” é o valor padrão para o argumento “Critério de parada”. A comunicação com o Engine R ocorre através do nome “crit”, isto é, o programa acessará o dado digitado pelo usuário através da variável chamada “crit”.

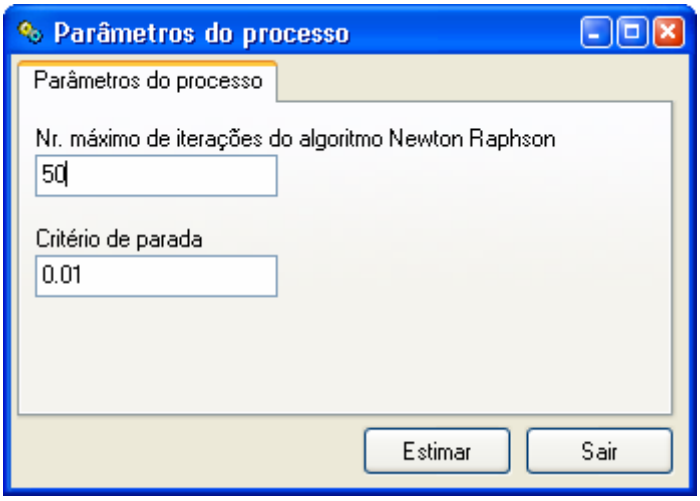

Figura 29 - Tela solicitando preenchimento dos argumentos exigidos pelo algoritmo

Importante observar que para definição de nomes de variáveis, deve-se obrigatoriamente obedecer às regras de criação de nomes de variáveis impostas pelo R.

5.4 Plug-ins da Categoria Geral

Os plug-ins da categoria Geral permitem adicionar funcionalidades adicionais ao ambiente e foi utilizado para a construção de Assistentes da Teoria da Resposta ao Item.

Os plug-ins desta categoria devem ficar armazenados num diretório denominado "plug-ins", que está na mesma pasta do programa executável. Um arquivo chamado index.xml deve necessariamente descrever o conteúdo o plug-in. A descrição do arquivo index.xml no formato XML Schema encontra-se no Anexo 3.

O arquivo index.xml dos plug-ins da categoria Geral fornecem as seguintes informações:

- Nome do plug-in
- Nome do grupo onde o plug-in deve ser exibido;
- Nome do programa principal
- Nome do arquivo de ajuda
- Ícone a ser utilizado para exibir o atalho do programa.

Um exemplo pode ser visto no Quadro 5**Erro! Fonte de referência não encontrada.**.

Quadro 5 - Exemplo de declaração de plug-in da categoria Geral

```
<plugin>
  <name>4 - Escore verdadeiro</name>
```

```
   <page>Básico TRI</basico>
   <program>p04.r</program>
   <hint>doc.txt</hint>
   <icon>icone.bmp</icon>
</plugin>
```

6 RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados obtidos no desenvolvimento do presente trabalho. Em seguida serão discutidos detalhes e características de cada um dos resultados.

6.1 Resultados

Foi desenvolvido um ambiente gráfico em Delphi 2007. Todos os arquivos fontes deste software estão disponíveis em http://www.inf.ufsc.br/~dandrade/ambientetri.

Uma implementação do modelo de 3 parâmetros desenvolvido em R por (PINHEIRO, 2006) foi adaptada para utilizar o modelo para construção de métodos da TRI especificado neste trabalho. Os seguintes algoritmos e ferramentas da TRI da implementação original foram ajustadas à especificação proposta:

- Estimação dos parâmetros dos itens;
- Estimação da variável latente
- Curva característica do item
- Curva característica do teste
- Função de informação do item
- Função de informação do teste
- Calculo de escore verdadeiro
- Equalização de teste

Para efetuar esta alteração, foram removidas todas as variáveis globais que estavam em uso. Os parâmetros exigidos por cada programa foram explicitados através da interface em arquivo XML.

Utilizando uma base de dados de teste, obtida em (PINHEIRO, 2006), foram processados os cálculos de estimação dos parâmetros dos itens e da variável latente. Em seguida, foi possível fazer uso das ferramentas da TRI para avaliar os resultados. Isto é, foi possível reproduzir os mesmos resultados da implementação original.

6.2 Discussão

O modelo proposto neste trabalho para construção de métodos da Teoria da Resposta ao Item é dependente exclusivamente da linguagem R, o que torna os pacotes de métodos (plug-ins) construídos segundo este modelo aplicáveis em qualquer

ambiente que seja capaz de reconhecer as interfaces dos programas e que implemente um mecanismo para comunicação com o Engine R. Ou seja, é possível construir novos ambientes para lidar com a Teoria da Resposta ao Item e reutilizar todos os pacotes de métodos da TRI que sejam desenvolvidos segundo o modelo proposto.

A especificação proposta admite a evolução do modelo, pois há uma propriedade que é utilizada para indicar a versão da especificação. Desta forma, é possível que o ambiente de software que faça uso dos pacotes lide com especificações de diferentes versões. A versão do modelo proposto neste trabalho é a versão 1.0.

O modelo favorece o encapsulamento do código ao promover a explicitação dos parâmetros dos programas. Entretanto, mesmo que não seja recomendável, o pesquisador - desenvolvedor de novos métodos da TRI – ainda tem liberdade de criar variáveis globais.

O fato da função que lê e interpreta o arquivo de dados de entrada estar disponível através de um programa R, permite que novos formatos de arquivos de dados sejam desenvolvidos, sem ser necessário recompilar o ambiente gráfico que suporta os plug-ins da TRI.

O uso da linguagem R permite aproveitar o conhecimento já existente dos pesquisadores nesta linguagem, visto que R é uma linguagem popular entre os pesquisadores da TRI. Acredita-se que o esforço em construir novos métodos utilizando o modelo deste trabalho seja baixo.

Para auxiliar a depuração de programas escritos em R, o ambiente desenvolvido neste trabalho se encarrega de salvar o contexto do Engine R antes e depois de executar os programas de estimação.

7 CONCLUSÕES E TRABALHOS FUTUROS

Foi definido um modelo (especificação) para construção de métodos da TRI. A definição de um modelo para implementação de métodos da TRI uniformiza o modo como os pesquisadores constroem novos métodos da TRI. O modelo requer que os programas sejam escritos em linguagem R e as interfaces dos programas sejam descritas em arquivos no formato XML. O empacotamento dos programas R e interfaces XML constituem um plug-in de método da TRI. Este formato de empacotamento de métodos da TRI deve facilitar a distribuição e compartilhamento de implementações com outros cientistas.

Um ambiente voltado para uso da TRI foi apresentado. O ambiente é capaz de suportar os plug-ins de métodos da TRI que seguem a especificação deste trabalho. Para que o ambiente gráfico seja capaz de interpretar programas R foi necessário utilizar um mecanismo para estabelecer comunicação com o Engine R. O mecanismo utilizado, denominado RD-COM, baseia-se na tecnologia COM.

O ambiente gráfico permite que o usuário realize análises de testes utilizando a TRI sem ter que recorrer aos programas fontes para compreender quais informações são indispensáveis para processar os algoritmos de estimação e a execução de ferramentas da TRI. Portanto, o ambiente permite que os usuários usufruam da TRI com maior facilidade. Ainda assim, um estudo da teoria torna-se necessário para que o usuário possa compreender os gráficos e relatórios fornecidos pelos métodos.

Como sugestão para trabalhos futuros, pode-se citar:

- desenvolver um ambiente (*front-ent*) em outra plataforma de software;
- construir um ambiente voltado para atender a análise de testes educacionais, utilizando termos e expressões conhecidas da área da educação e fazendo uso de assistentes (*wizards*);
- criação de assistentes para orientar o pesquisador na criação das interfaces dos programas e no empacotamento dos mesmos;
- permitir a partir do próprio ambiente gráfico efetuar download de plug-ins da TRI;
- incorporar ao ambiente os pacotes já desenvolvidos em R para a TRI;

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMS, Marc. **PSE Research at Virginia Tech: An Overview**. Disponível em http://research.cs.vt.edu/pse/intro.html. Acesso em Novembro de 2006.

ANDRADE, Dalton Francisco de; TAVARES, Héliton Ribeiro; VALLE, Raquel da Cunha. **Teoria da Resposta ao Item**: Conceitos e Aplicações. São Paulo: Associação Brasileira de Estatística, 2000.

ARNOLD, Dorian C.; DONGARRA, Jack. **Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications**. In: Working Conference on the Architecture of Scientific Software. Ottawa, Canadá: Proceedings of the IFIP TC2/WG2.5. 2000, Vol. 188, p. 39-56.

Assessment Systems Corporation. XCALIBRE - Marginal Maximum-Likelihood Estimation, Versão 1.10. 2007.

BAKER, Frank**. The Basics of Item Response Theory**. EUA: ERIC Clearinghouse on Assessment and Evaluation, 2001, 2a. Edição.

BRASIL. Ministério da Agricultura, Pecuária e Abastecimento. **Repositório AgroLivre: software científico: SOC: informações de projeto**. Brasilia, DF: MAPA: Embrapa, 2007. Disponível em: http://repositorio.agrolivre.gov.br/projects/soc/. Acesso em 22 ago. 2007.

BOISVERT, Ronald F. **The Architecture of Scientific Software**. In: Working Conference on the Architecture of Scientific Software. Ottawa, Canadá: 2000, p. 351-356.

CHAMBERS, John. **Programming with Data**: A Guide to the S Language. Berlim, Alemanha: Springer-Verlag, 1998.

CHATLEY, Robert. **Predictable Dynamic Plugin Architectures**. Tese (Doutorado) - Universidade de Londres, 2005.

CHEN, Chih-Ming; LEE, Hahn-Ming; CHEN, Ya-Hui. **Personalized e-learning system using Item Response Theory**. Computers & Education . 2005, Vol. 44, p. 237-255. ISSN: 0360-1315.

Codegear. **Delphi, Versão 2007**. 2007.

CRNKOVIC, Ivica; SCHMIDT, Heinz G.; HEINEMAN, George T.; et al. **Component-Based Software Engineering**. Springer-Verlag Berlin, 2007.

DEPRINCE JR, Wayne; HOFMEISTER, Christine. **Analyzing Commercial Component Models**. In: 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance. IFIP 17th World Computer Congress. 2002, p. 205-219.

DONGARRA, Jack; SULLIVAN, Francis. **Top Ten Algorithms of the Century**. IEEE Educational Activities Department, 2000, Vol. 2, p. 22-23.

ETTER, Delores M. **Engineering Problem Solving with Matlab**. EUA: Prentice Hall, 2006, 2a. Edição.

FRITZON, Peter. **Principles of Object-Oriented Modeling and Simulation with Modelica 2.1**. Wiley-IEEE Press, 2004.

GALLOPOULOS, Efstratios; HOURTIS, Elias N.; BRAMLEY, Randall; et al. **Problem-Solving Environments for Computational Science**. IEEE Computational Science & Engineering . Los Alamitos, EUA: 1997, Vol. 4. ISSN: 1070-9924.

GALLOPOULOS, Efstratios; HOURTIS, Elias N.; RICE, John R. **Workshop on problem-solving environments**: findings and recommendations. Nova Iorque, EUA: 1995, Vol. 27, p. 277-279.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; et al. **Design Patterns**: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995.

GERSHON, Richard; CELLA, David; DINEEN, Kelly; et al. **Item response theory and health-related quality of life in cancer**. Expert Review of Pharmacoeconomics and Outcomes Research. Future Drugs, 2003, Vol. 3, p. 783-791.

GOEL, A.; PHANOURIOU, C.; KAMKE, F.; et al. WBCSim: **A Prototype Problem Solving Environment for Wood-Based Composites Simulations**. 1999, Vol. 15, p. 198-210.

Insightful. S-Plus, Versão 8.0. 2007.

Jurgen A Doornik. Ox, Versão 5.0. 2007.

HALE, Dave. **Plataforms for Open-Source Scientific Software**. In: 68th EAGE Conference & Exhibition incorporating SPE Europec 2006. Vienna, Austria: 2006.

HAMBLETON, Ronald K; SWAMINATHAN, H.; ROGERS, H. Jane. **Fundamentals of Item Response Theory**. North Caroline: Sage Publications, 1991.

HARVEY, Robert J.; HAMMER, Allen L. **Item Response Theory**. The Counseling Psychologist. 1999, Vol. 27, Nr. 3.

HEATH, Michael T. **Scientific Computing: An Introductory Survey**. Nova Iorque, EUA: WCB/McGraw-Hill, 2002.

HERZUM, Peter. **Component Factory**: A Comprehensive Overview of Component-Based Development for the Enterprise. Wiley, 1999.

HEY, Anthony J. G.; PAPAY, J.; KEANE, A. J.; et al. **Component Based Problem Solving Environment**. In: 8th International Euro-Par Conference on Parallel Processing. Londres, Inglaterra: Lecture Notes In Computer Science. 2002, Vol. 2400, p. 105-112.

HILL, Cheryl D.; LANGER, Michelle M. **PlotIRT**: A Collection of R Functions to Plot Curves Associates with Item Response Theory. Applied Psychological Measurement. SAGE Plublications, 2007, Vol. 31, No. 5.

HOUSTIS, Elias N.. **The Role of Problem Solving Environments in Engineering and Mathematics Education.** Resolución de Problemas en la Educación Matemática. 2003.

HYARIC, Antoine Le. **List of Free Numerical Analysis Software.** Disponível em: http://www.ann.jussieu.fr/free.htm. Acesso em Novembro de 2007.

KUNCICKY, David. **Matlab Programming**. Prentice-Hall, 2003.

LAU, Kung-Kiu; WANG, Z. **Software Component Models**. IEEE Transactions on Software Engineering. 2007, Vol 33.

LUJÁN, Mikel. **Building an object oriented problem solving environment for the parallel numerical solution of PDEs**. In: Conference on Object-Oriented Programming, Systems, Languages, and Applications. Nova Iorque, EUA: ACM Press, 2000.

MAHMOUD, Qusay H.. **Distributed Programming with Java**. Manning Publications, 1999.

MAINDONALD, John; BRAUN, John. **Data Analysis and Graphics Using R**. Cambridge University Press, 2003.

Maplesoft. Maple, versão 11. 2007.

MathWorks. Matlab - Matrix Laboratory, versão 2007b. 2007.

MAYER, Johannes; MELZER, Ingo; SCHWEIGGERT, Franz. **Lightweight Plug-In-Based Application Development**. In: International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World. 2002.

MULTILOG, Versão 7.0. [S.I]: Scientific Software International, 2007.

MORRISON, Michael. **Presenting Javabens**. Sams Publishing, 1997.

O'LEARY, Dianne P. **Guidelines on Writing Scientific Computing Software**. 2005.

PASQUALI, Luiz. **Psicometria**: Teoria dos Testes na Psicologia e Na Educação. Editora Vozes, 2003.

PINHEIRO, Conrad Elber. **Implementação de Métodos Estatísticos para Avaliação Educacional no Software R**. Dissertação (Mestrado) - IME. USP, 2006.

PIRONNEAU, Oliver. **Prospectives for Scientific Computing**. In: Science and Industry Advance with Mathematics (SIAM 2000). Japão: 2000.

Purdue Problem Solving Environments Research Group. **Problem Solving Environments Projects, Products, Applications and Tools**. Disponível em: http://www.cs.purdue.edu/research/cse/pses/research.html. Acesso em 15 de Dezembro de 2006.

R. **The R Foundation for Statistical Computing Version 2.2.0**. Software livre. Disponível em: www.r-project.org, 2006.

RANNA, Omer F.; SHIELDS, Matthew. **A Collaborative Code Development Environment for Computational Electro-Magnetics**. In: Working Conference on the Architecture of Scientific Software. Ottawa, Canadá: IFIP TC2/WG2.5. 2000.

REEVE, Bryce B.; FAYERS, Peter. **Applying item response theory modelling for evaluating questionnaire item and scale properties**. Oxford, 2005, p. 55-73.

RICCIONI, Paulo Roberto. **Introdução a Objetos Distribuídos com CORBA**. Visual Books, 2000.

RICE, John R.; BOISVERT, Ronald F.. **From Scientific Software Libraries to Problem-Solving Environments**. IEEE Computational Science & Engineering. IEEE Computer Society Press, 1996, Vol. 3.

RUDNER, M. Laurence. **PARAM-3PL - Calibration Software for the 3 Parameter Logistic IRT Model**. Disponível em: http://edres.org/irt/param/, 2005.

RIZOPOULOS, Dimitris. **ltm: An R package for latent variable modelling and item response theory analyses**. Journal of Statistical Software. 2006, Vol. 17, p. 1-25.

SSI - Scientific Software International. Parscale, Versão 4.0. 2007.

SSI - Scientific Software International. Bilog-MG, Versão 3.0. 2007.

SYMBOLICS. Macsyma, Versão 2.4. 1995.

TREFETHEN, Anne E.; MENON, Vijay S.; CHANG, Chi-chao; et al. **MultiMATLAB**: MATLAB on Multiple Processors. 1996.

VENABLES, W. N e SMITH, M. **An introduction to R**. Disponível em http://cran.r-project.org/doc/manuals/R-intro.pdf. Acesso em 12 de Junho de 2006.

VÖLTER, Markus. **PluggableComponent** – A Pattern For Interactive System Configuration. In: 4th European Conferente of Pattern Languages of Programming and Computing. 1999

WOLFRAM Mathematica. Mathematica, Versão 6.0. 2006.
.
WONG, Kam-Fai; LYU, Michael R.; CAI, Xia. **Component-Based Software Engineering**: Technologies, Development Frameworks, and Quality Assurance Schemes. In: 17th Asia-Pacific Software Engineering Conference. 2000.

ANEXO 1

Descrição no formato XML Schema do arquivo XML usado para declarar o conteúdo de um plug-in de métodos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="method">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="parameters"/>
        <xs:element ref="abilities"/>
        <xs:element ref="equating"/>
        <xs:element ref="output"/>
        <xs:element ref="version"/>
        <xs:element ref="modelName"/>
        <xs:element ref="description"/>
        <xs:element ref="compability"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="parameters">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="program"/>
        <xs:element ref="input"/>
        <xs:element maxOccurs="unbounded" ref="output"/>
        <xs:element maxOccurs="unbounded" ref="name"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="abilities">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="program"/>
        <xs:element ref="input"/>
        <xs:element ref="output"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="equating">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="program"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="compability" type="xs:decimal"/>
  <xs:element name="version" type="xs:decimal"/>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="modelName" type="xs:string"/>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="program" type="xs:string "/>
  <xs:element name="input" type="xs:string "/>
  <xs:element name="output">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="program"/>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

ANEXO 2

Descrição no formato XML Schema do arquivo XML usado para declarar o a interface de um programa R.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="variables">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="variable"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="variable">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="label"/>
        <xs:element ref="default"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="label" type="xs:string"/>
  <xs:element name="default" type="xs:string"/>
</xs:schema>
```

ANEXO 3

Descrição no formato XML Schema do arquivo XML usado para declarar um plug-in da categoria Geral.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="plugin">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="page"/>
        <xs:element ref="program"/>
        <xs:element ref="hint"/>
        <xs:element ref="icon"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="page" type="xs:string"/>
  <xs:element name="program" type="xs:string"/>
  <xs:element name="hint" type="xs:string"/>
  <xs:element name="icon" type="xs:string "/>
</xs:schema>
```